AFIT/GE/ENG/92D - 33

Automated Face Recognition System

THESIS

Kenneth R. Runyon
Captain, USAF

AFIT/GE/ENG/92D -33

Approved for public release; distribution unlimited

AFIT/GE/ENG/92D-33

Automated Face Recognition System

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

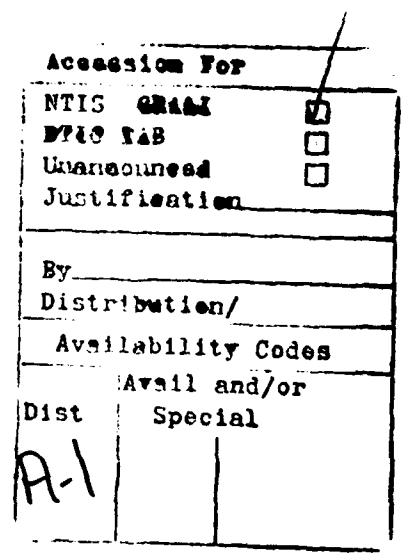
Requirements for the Degree of

Master of Science in Electrical Engineering

Kenneth R. Runyon, B.S.S.E.

Captain, USAF

December, 1992



Acknowledgments

I would like to thank my thesis advisor, Dr. Steve Rogers, and the rest of my thesis committee, Dr. Dennis Ruck, Dr. Matthew Kabrisky, and Dr. Mark Oxley. Their help made this thesis the educational experience I had hoped for.

I would like to thank the *Face Guys*, Capt Kevin Gay and Capt Dennis Krepp. This assignment might have been tough without them. As it was, I think I actually enjoyed myself. I also wish to thank Dan Zambon and Dave Doak for their tireless efforts in administering the computer network.

Finally, I want to thank my family; my girls, Anna and Sarah, who have made significant sacrifices in their short lives, and most especially, my lovely wife Lisa. If I have ever succeeded, it is because of her.

Kenneth R. Runyon

Table of Contents

List of Figures

Abstract

In this thesis three variations of an end-to-end face recognition prototype system are developed, implemented and tested. Each version includes real-time image collection, automated segmentation, preprocessing, feature extraction, and classification. The first version uses a Karhunen Loéve Transform (KLT) feature extractor and a K-nearest neighbor classifier. Version two uses the same feature set but utilizes a multilayer perceptron neural network with a back propagation learning rule. Finally the third version uses a Discrete Cosine Transform as the feature extractor and the K-nearest neighbor as the classifier. Only the KLT versions of the system were tested.

The tests were based on three image sets, each collected over multiple days to analyze the effect on recognition accuracy of variations in both the image collection environment and the subjects over time. The first set consisted of 23 Subjects and was taken over a two day period. The second set consisted of four users and was taken over a seven day period. Finally, the third set consisted of 100 images of a single subject collected over several weeks.

The K-nearest neighbor achieved the following scores:

- 67% for a 23 Subject - 2 Day Test
- 90% for a 4 Subject - 7 Day Test
- 69% for a single subject - Long Term Test

The Multilayer Perceptron had the following recognition accuracies:

- 74% for a 23 Subject - 2 Day Test
- 100% for a 4 Subject - 7 Day Test
- 100% for a single subject - Long Term Test

Automated Face Recognition System

I. Problem Description

1.1 Introduction

This thesis discusses the design, implementation, and testing of an end-to-end, automated, face recognition system. This prototype is being developed in response to a DoD requirement for such a system to provide controlled access to computer accounts. While pieces of the technology exist in a mature enough form to be immediately implemented, other algorithms must be developed and integrated to fully establish the system.

An automated face recognition system is a computer application which identifies individuals by analyzing a video input of the person's face. The basic components which typically make up such a system are a visual sensing system to collect the image, a segmentation algorithm to discount variations of the background, the feature extraction algorithm to pull out the unique characteristics of the image that distinguish an individual from anyone else, and a classifier to decide which individual the image represents.

Automated face recognition machines will allow the computer to assume a new and very diverse role in society. Its application will range in function from establishing a human-computer interface for disabled children to spotting terrorists in a crowded airport(17). Any system which currently requires passwords or personal identification numbers (PINS) will eventually provide access to the desired information or service based on visual authentication of the user as a secondary or even primary level

of security. Such systems include automated teller machines, computer accounts, and even the security guard whose function is to compare the photos on restricted entry badges to the face of the wearer. The advantage of an automated system is that we can eliminate the need to subject a human to mundane, repetitive tasks and do away with inconvenient passwords and PINs which are easily lost, stolen, or forgotten. Another use of such systems is to provide a new type of interface between man and machine. The capability for computers to interface efficiently with humans in the same way that humans interface with each other (verbal and nonverbal communication) is still very much a research topic; however, the technology is mature enough to provide a human/computer interface to physically disabled persons whose controllable motor skills have been reduced to facial expressions(17).

The problem description begins with an overview of the face recognition research being conducted at the Air Force Institute of Technology. The capabilities and shortcomings as well as the current state of that system are addressed. The problem statement is then given, and the research objectives stated. Assumptions are then identified and the scope and limitations of the effort given. A statement of standards is then made, followed by a discussion of the approach used to study this problem. The chapter concludes with an overview of the remaining chapters of this thesis and a summary.

1.2 Background

The development of an automated face recognition system has been a topic of research at the Air Force Institute of Technology since 1985. This research resulted in the Autonomous Face Recognition Machine (AFRM) which was refined and improved over a number of thesis cycles(14, 16, 9, 15, 11). The system was originally developed to verify Routh's Cortical Thought Theory (CTT)(12). While CTT faded into obscurity, the AFRM was continued for several thesis efforts. The actual AFRM system no longer exists in the form of hardware and software; however the documentation from that system

will be used to guide this thesis effort and to provide a means of gauging the results. The following sections list the students who have contributed to the AFRM and the particular achievements of their research. Other face recognition efforts will be discussed in chapter Two.

1.2.1 **Russell** was the originator of the AFIT face recognition effort. His system was a somewhat manual process which mapped faces to a two-dimensional "gestalt" vector. This vector was compared by the nearest neighbor algorithm to vectors stored in a data base(14).

1.2.2 **Smith** added automatic face location and windowing algorithms to Russell's face recognition system which allowed the system to work autonomously. Thereafter the system was commonly referred to by the AFRM acronym (Autonomous Face Recognition Machine)(16).

1.2.3 **Lambert** considerably improved the segmentation algorithm used by the AFRM. He reduced the recognition time to seconds. His segmentation technique was based on a frame subtraction algorithm to detect motion. Once motion was detected, a process dubbed "Lambertization" was used on the image to discount non-uniform luminance. An algorithm to determine if the segmented image contained a face was run before passing the image over to the feature extractor(9).

1.2.4 **Sander** replaced the "gestalt" feature vector with a feature vector comprised of the coefficients of the discrete Fourier transform (DFT) of the image. His system used the DC component and the first two harmonics of the DFT. A second contribution was the addition of a back propagation neural net as the classifier mechanism(15).

1.2.5 **Robb** added another harmonic to the DFT feature vector, fixed various bugs in the original code, and thoroughly documented the system. This final version of the AFRM achieved a

recognition accuracy of 73 percent for a training set of 45 subjects. Her thesis provides the best and final documentation for the AFRM(11).

1.2.6 **Current AFRM Research** The AFRM was successful from the standpoint that it proved the ability of a computer to detect and recognize an individual. Unfortunately, the 73 percent recognition rate of the system is too low to be of practical use. Thus in the latest thesis cycle, Suarez and Goble set out to increase the recognition accuracy of the system by exploring orthogonal feature sets. Suarez explored the Karhunen Loéve Transform(17) while Goble researched the Discrete Cosine Transform(5). These algorithms are used primarily in the data transmission arena as a compression scheme to reduce images for efficient transmission while retaining the unique information necessary to reconstruct the image at the receiving end. Both efforts resulted in recognition accuracies which were superior to the DFT. The goal of this thesis is to implement each of these algorithms into an end-to-end, face recognition system.

1.3 Problem Statement

The AFRM yields recognition results which are inadequate for operational systems. New algorithms have been developed to overcome this problem. These algorithms will be implemented into an end-to-end, face recognition system to meet a DoD requirement. This system will be used in an application to provide controlled access to computer accounts by visually authenticating prospective users.

1.4 Research Objectives

The objectives of this research are :

- Collect a database of test images.
- Design and implement an end-to-end face recognition system based on Suarez's Karhunen Loéve Transform feature set and the nearest neighbor classifier(17).
- Evolve the nearest neighbor classifier routine into a K-nearest neighbor classifier.
- Implement a version of the face recogntion system using the Discrete Cosine Transform feature set developed by Goble(5).
- Implement a KLT version of the face recognition system with a backpropagation neural network classifier.
- Provide recognition accuracy benchmarks for the resulting face recognition system.
- Invesitigate the effect of using Gay's(4) automated segmentation technique on overall recogntion accuracy.
- Compare the results of a neural network classifier against that of the K-nearest neighbor classifier.
- Investigate the problems and solutions to training and testing on images collected over several days.

1.5 Assumptions

- The entire system must be implementable using the resources available on a Sun Microsystems SPARCstation.
- The user attempting to login into the system is cooperative.

- The only motion in front of the camera is the user's face.
- Recognition of faces will be limited to frontal views.

1.6 Scope and Limitations

The scope of this thesis is to study the performance of a prototype face recognition system. All conclusions are based on empirical evidence.

1.7 Standards

The performance criteria for all of the techniques is classification accuracy, speed and user friendliness. Accuracy is the most important criteria; though speed is high on the list of desired characteristics. User friendliness will be implemented as much as possible given time and resource constraints. The accuracy of classification of each technique is the ratio of correct classifications to the total number of test data.

1.8 Approach and Methodology

As a part of this thesis, a software application is developed on the Sun Microsystems SPARC-station. This application combines segments of existing software with new programs written in ANSI C. The algorithms allow the user to collect real time test and training images, train the system using those images, and perform recognition of potential users. All software is documented in Appendix A.

1.9 Thesis Overview

Chapter Two presents a review of current literature related to face recognition systems. Chapter Three provides a detailed description of the methodology. Chapter Four provides a description of the test results based on the methodology. Chapter Five presents conclusions based on the test results in Chapter Four.

1.10 Summary

This thesis developes an overall system to be used as both a baseline and a testbed for future efforts in developing face recognition algorithms. The system performs image collection, segmentation, feature extraction, training, and recognition.

II. Literature Review

2.1 Introduction

This literature review discusses the development and implementation of face recognition systems. A typical face recognition process includes algorithms to perform segmentation, feature extraction, and classification. Segmentation is the process of separating the face (or head) of the individual to be recognized from the clutter of the background. Feature extraction is a process which reduces the dimensionality of the input image while maintaining the information necessary to uniquely distinguish an individual. Classification is the algorithm which examines a given feature vector and "recognizes" the individual in the input image.

During the last decade, several major institutions have invested considerable resources in designing and testing such systems(6, 10, 18, 1). While various levels of success have been achieved, research continues in an effort to find better algorithms to improve all phases of the recognition process. The goal is to design a system with an acceptable recognition accuracy that is robust enough to operate in real world conditions.

The DoD sponsor for this thesis desires a system which can operate within the computational constraints of a Sun SPARCstation. Their objective is to use this system as a means to prevent unauthorized access to computer accounts. Therefore the scope of this review will be limited to systems which make use of accurate, robust, and computationally efficient algorithms.

The review begins with a discussion of the WISARD system developed at Brunel University in the United Kingdom(6). Next a summary of the back propagation neural network effort being conducted at Los Alamos National Laboratory will be given(10). Then the Karhunen Loéve Transform

technique being used at the Massachusetts Institute of Technology will be discussed(19, 18). After that a description of the identity neural network research being done at the University of California San Diego will be given(3, 1). Finally, the review will end with a short overview of the Karhunen Loéve and Discrete Cosine Transform feature set research which was accomplished at the Air Force Institute of Technology(17, 5).

2.2 *WISARD*

A face recognition system based on neural networks has been developed by a team of researchers at Brunel University in the United Kingdom(6). This system is unique in that the perceptron style neural network architecture is implemented in hardware using a VLSI random access memory (RAM). The 153 x 214 pixel input images are first binarized and then the pixels are collected into groupings of four where each of the four pixels in the group was randomly selected from all 32,742 pixels available in the image. The four pixel grouping is a feature referred to as a tuple. There are 8,185 tuples in a given input image. The tuple groupings make the feature set sensitive to global patterns in the image; though, this does not imply sensitivity to faceness. Each of the four bits for a given tuple is connected to the address lines of the RAM (see figure 2.1). The remaining address lines are connected to a binary counter which signifies which of the tuples is being input. The output of the RAM for a given tuple is either a one or zero. The 8,185 outputs for a given image are summed together to provide an overall recognition score. The network with the highest score indicates the subject reconized in the input image. The equivalent perceptron architecture is shown in figure 2.2. Using a single 1 mega-bit RAM, a network (or discriminator) of this configuration is implemented for each subject in the training set.

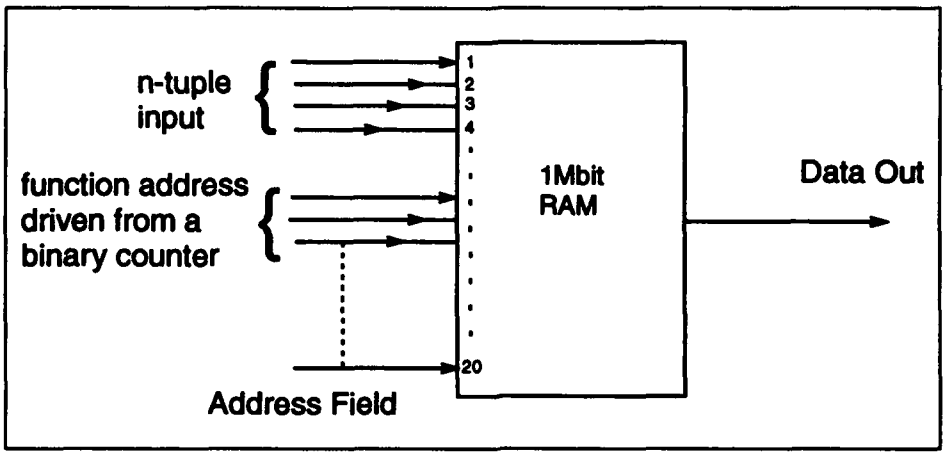


Figure 2.1. RAM Neural Network Implementation
(6)

Training the system consists of collecting an input image, grouping the image into tuples, applying the tuples and binary counter output to the address lines of the RAM, and writing a binary value of one into the accessed memory location. The training set for the system is gathered in real time and the training images are not stored. The training process is conducted on a train/test cycle by collecting an input image, training the system on the image, then testing the response of the network to a new input image of the same subject. This iterative process continues until the recognition score is in excess of 120 or 95 percent of the maximum response; regardless of the position of the subject or his expression (frontal views are required). On average, 200-400 images must be collected for each subject; however, at a processing rate of 25 images per second, the network is trained on each individual in less than 20 seconds. All training images are collected at the same sitting.

To perform recognition, the system collects a test face which is processed through each discriminator. The test images used in this particular test were collected during the training session, although the system is shown to work in real time as well. Classification is achieved by determining which

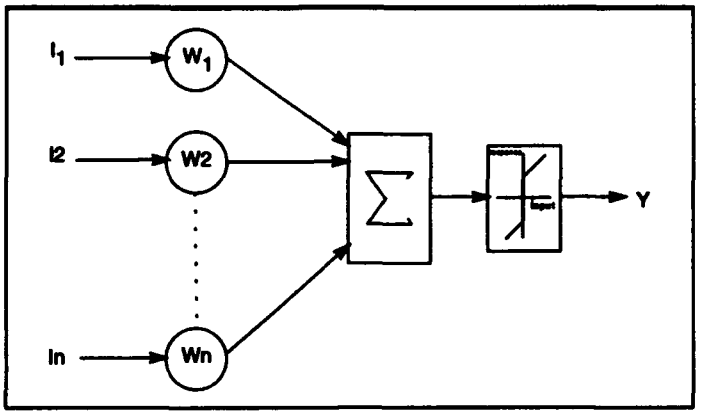


Figure 2.2. Equivalent Neural Network
(6)

discriminator gives the highest response for that input image. Using these techniques the system was trained for 16 subjects and achieved a 100 percent recognition accuracy(6).

A more industrial version of the system has been developed by Scientific Generics Ltd., of Cambridge(2). They have improved the system by adding a sophisticated infrared illumination system to the front end to overcome the variable effects of ambient illumination. In addition, a post processing system has been added to give the time and location of a face within the input scene. The system is built to provide accurate recognition at near real-time speeds.

2.3 Los Alamos National Laboratory

The Systems Concepts Analysis Team of the Space Sciences and Technology Division, Los Alamos National Laboratory is investigating using a backpropagation neural network as a face verification system. Their emphasis seems to be on how much and what kind of data should be collected to train a neural network to recognize in general(10).

They have collected over 11,416 images. Their original database started with 20 pictures each of 511 subjects. Recently the number of subjects in the database was increased by 249 to 760 individuals. The new subjects were primarily college students from Auburn. Four or five training images of each new subject were collected offline. All training images for each individual were collected within a one minute interval. Constraints on the data were constant lighting conditions and a neutral background. Subjects talked, changed expressions and ate cookies during the image grabbing process.

They implemented the backpropagation algorithm using a commercially available neural network design application. A single person verification network was implemented and trained with the following empirically derived parameters for each test subject:

- Input nodes: 1400 (raw pixels)
- Hidden nodes: 20
- Output nodes: 1 (verfication)
- Momentum: 0.50
- Hidden layer learning: 0.30
- Output layer learning: 0.15
- Random initialization of weights from -0.1 to 0.1 (uniformly distributed)
- Input vectors were rescaled to range from -1 to 1

The data was sorted according to demographic categories such as sex, skin tone and hair color. Various compositions of the data were used resulting in six different training sets. Some sets were carefully organized by demographics while others were constructed by randomly selecting from all subjects in the data base. Training sets were varied to include 5, 10, 15, and 20 percent of the data

base. The network was trained for 15,000 iterations then tested for accuracy. The errors were divided into two classes; false acceptances and false rejections. Images which fell between classifications were counted as "don't knows".

The network was trained most accurately using larger percentages of randomly drawn data. Error rates were minimal when 20 percent of the data was drawn randomly for training. Using this training set false acceptance was found to be 0.02 percent while false rejection was 8.7 percent (10).

2.4 Massachusetts Institute of Technology

Turk and Pentland have developed a system which locates and recognizes faces in near-real-time. Their system consists of a video camera, an image processor board, a Sun 3/160 dedicated to motion analysis, and a second Sun SPARCstation which performs the recognition process; though both algorithms could be executed on one computer.

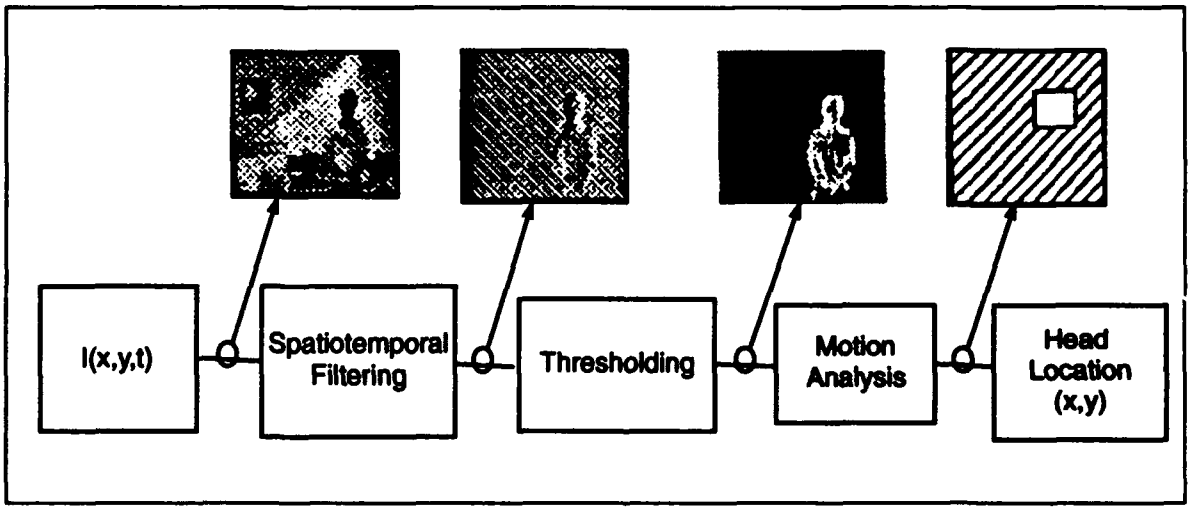


Figure 2.3. MIT Face Recognition System
(18)

Segmentation of the head from the background is accomplished by the process shown in figure 2.3. Motion detection is accomplished by a standard frame subtraction algorithm. The motion is tracked over time to provide input to the motion analysis algorithms. The motion analysis program applies heuristics to the detected motion using simple rules such as "the head is the smaller blob on top of the larger blob (body)" and "the head must move in a contiguous motion (doesn't disappear at one point and reappear in a new location)". Once a head image is found a subimage is segmented out of the original picture and sent to the recognition routines. The motion image is also used to provide an estimate of scale in that the size of the blob that is assumed to be the moving head determines the size of the subimage to send to the recognition stage. The subimage is rescaled to match the dimensions (number of pixels in row and columns) of the eigenfaces. While this technique works reasonably well, Turk proposes that the eigenfaces be scaled to several sizes to produce a system which is more robust with regard to scale(18). Head tilt is also a problem which has been addressed by using symmetric patterns for frontal views, to develop symmetry operators which estimate head orientation. The image can then be rotated to align the head; however, performance is marginal because the algorithm tends to make all head objects circular.

Eigenfaces are the set of basis functions which span and define a face space. This basis set which is optimally tuned to the training data is derived using the Karhunen Loéve principal component analysis (7). Face images can then be transformed to face space by expressing them as a linear combination of the eigenfaces. Thus an entire set of large dimensional images can be reduced to a very low number of coefficients. The coefficients for each training image and the identity of the individual in the image are stored in a data file for future recognition. Prototypes are taken at 15 degree increments to make the system more robust.

Recognition is carried out using a nearest neighbor classifier. Candidate faces detected by the motion system are transformed to face space where the coefficients of the input face are compared by Euclidean distance to each face in the training set. The recognition system runs in a loop and outputs an image every time a face is recognized, which could be as much as two or three times per second.

Limited testing has been accomplished using a training set of four subjects with two prototypes for each subject. Three KL coefficients were used as the feature set. The test set consisted of seven images, two images of subjects with images in the training set, three images of subjects who were not included in the training set, a filtered and subsampled version of one of the original training images, and a noisy version of one of the original training images. The classifier was set with a threshold of 20.0. The system correctly classified the two test images with corresponding training images. It correctly classified the unknown images as unknown (in that they fell below the 20.0 threshold). The filtered image was correctly classified despite the effects of additional processing; however the noisy image was pushed outside the threshold for correct classification.

2.5 *University of California San Diego*

Garrison Cottrell of the Institute of Neural Computation UCSD has investigated a face recognition network which performs feature extraction and classification using back propagation neural networks(1). As with Turk and Pentland, Cottrell's feature set is holistic consisting of a basis set of images he refers to as holons. These images are very similar in appearance to Turk's eigenfaces (see figure 2.4). To extract these features, Cottrell passes manually collected 64 x 64 pixel images through a two layer auto-associator network consisting of 4096 input nodes, one for each pixel in the input image, 40 hidden layer nodes, and 4096 output nodes (see figure 2.5). The function of the auto-associator is to reproduce the same image at the output as that received at the input for all images in the training set.

Once the hidden layer is trained the output layer is removed and the response of the original hidden layer is input into a single layer backpropagation network. This new network is configured with an output node for each subject in the training set. The network is then trained on the training set using the original hidden layer weights derived from the previous auto-associative network. Once the network is trained, images are recognized by putting them into this new network and observing the output node which gives the maximum response. The subject which corresponds to this output node is the person in the test image.

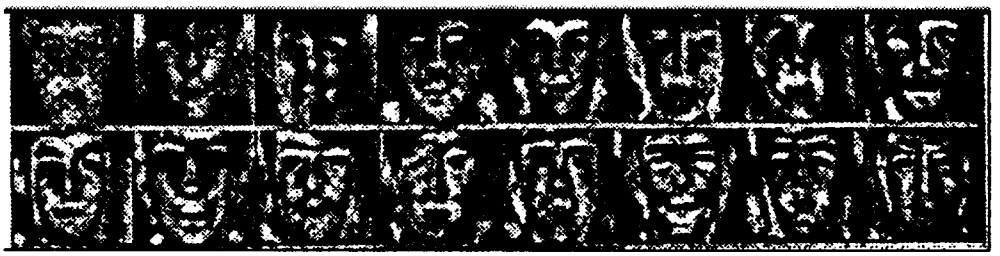


Figure 2.4. Identity Net Holons
(3)

Using this technique Cottrell trained the network with images of 10 males and 10 females. All subjects were UCSD college students. Images were manually centered and brightness as well as variance was normalized before training began. No mention is made of how many images of each person were actually trained on; however, each individual was asked to feign 8 different emotional expressions as data for additional tests concerning recognition of facial expression. Thus the training set consisted of at least 160 faces. The recognition accuracy for the system was 99 percent for all faces in the test set.

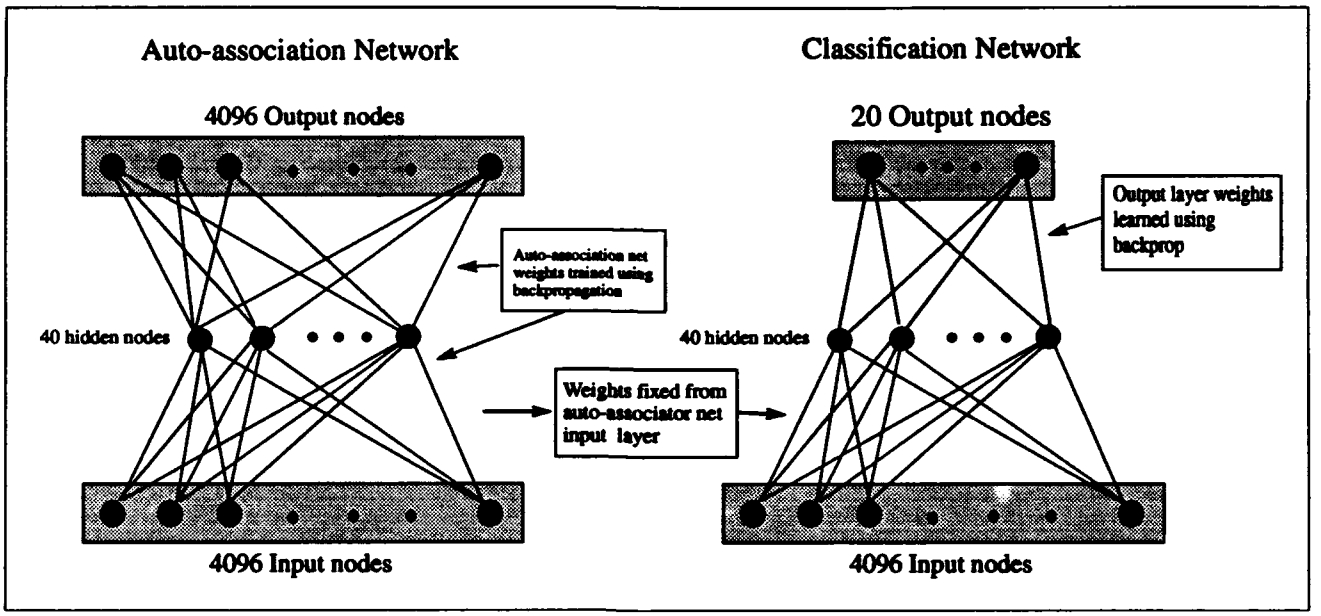


Figure 2.5. UCSD Recognition System
(8)

2.6 *Air Force Institute of Technology*

As stated previously, two researchers at AFIT collaborated on their thesis efforts to jointly investigate orthogonal feature sets. Suarez investigated the Karhunen Loéve Transform (KLT) while Goble researched the Discrete Cosine Transform (DCT). They worked together to develop the supporting software necessary to test their feature extraction algorithms. Consequently, a very good foundation for an end-to-end face recognition system was produced. The algorithms they developed include preprocessing, feature extraction, and a nearest neighbor classifier in addition to each researcher's particular transform. They collected an image database of 55 subjects with four training and two test images for each subject. The 128 x 128 pixel images were manually collected in a very controlled environment using a commercially available frame grabbing application. In collecting the images, subjects sat in front of a neutral background at the same distance and looked directly into the camera.

All pictures of each subject were taken within minutes of each other under constant lighting conditions. The images were then preprocessed using a correlation algorithm first to center the subject in the picture; then a gaussian windowing routine was used to highlight the center of the image (facial features) and de-emphasize the outer edge of the picture (the background). After another run through the centering algorithm the image was then transformed by either KLT or DCT.

2.6.1 Karhunen Loeve Transform In Suarez's system, figure 2.6, the images would now be used by the KLT routine to create a basis set of images called eigenfaces. These images are then used by a reconstruction algorithm to calculate the coefficients of the training images in face space. Suarez tested the KLT feature set for single person verification and multiple person recognition.

For single person verification he trained a backpropagation neural network configured with five input nodes, two hidden layer nodes, and two output nodes. The training set consisted of two classes of images, targets and non-targets. 16 training images were used for each class. He extracted five coefficients for each image and used these features to train the network using the hold one out method to train and test. Using this method he achieved a 92 percent recognition rate for single person verification. The results do not show individual results for false acceptance and false rejection.

Suarez tested the KLT feature set for multiple person recognition using only the nearest neighbor classifier. For this test he used four training images for each of 55 test subjects. He extracted 16 coefficients for each image. The feature set was tested using two test images of each subject, collected on the same day as the training images. He achieved a 95 percent recognition accuracy for this test(17).

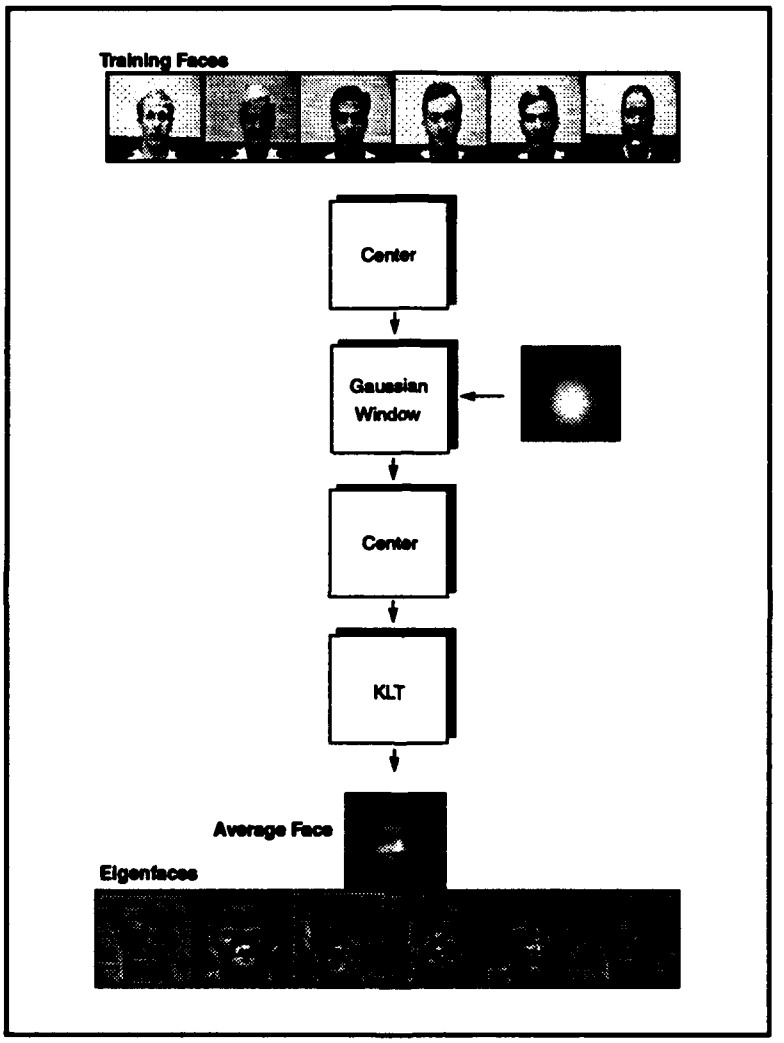


Figure 2.6. KLT Feature Extraction Process
(17)

2.6.2 Discrete Cosine Transform In Goble's system, figure 2.7, the images are directly transformed by the DCT software. The coefficients for each image, as well as the name of the subject in the image, are stored in a data file.

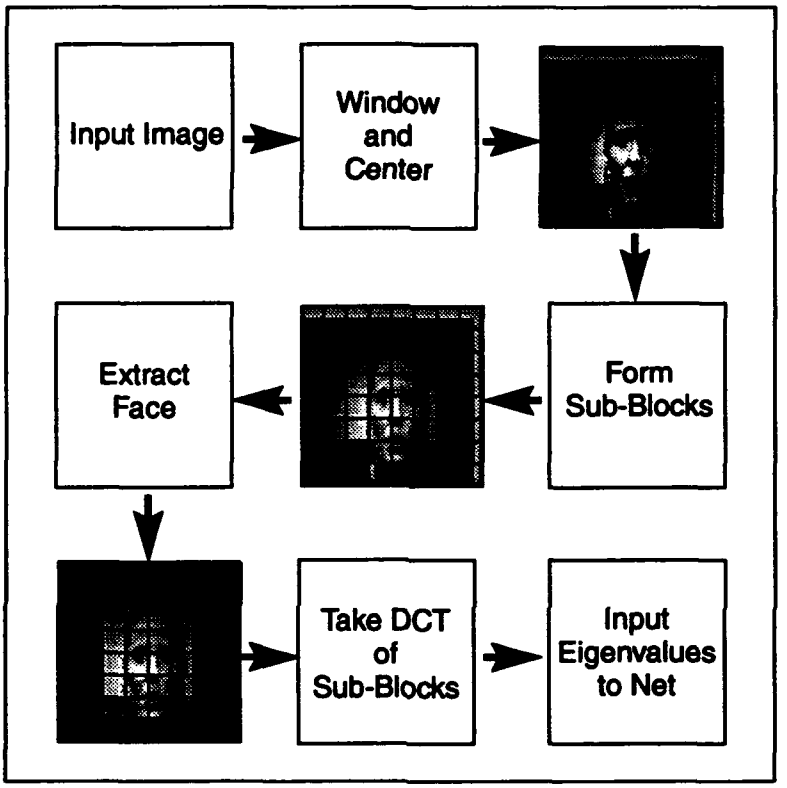


Figure 2.7. DCT Feature Extraction Process
(5)

Goble performed the same single person verification test using DCT as Suarez did using KLT. His training set also consisted of four images each of 55 people. The test set contained two images of each of 55 subjects. His feature set differed in that he used 7 DCT coefficients for the backpropagation neural net as opposed to Suarez's five coefficients. His false accept rate was 2 percent while his false reject rate was 3.5 percent.

For the multiple person recognition test Goble extracted 20 DCT coefficients from four training images of 55 different subjects. He tested using two test images of each subject and only used the nearest neighbor classifier. This resulted in a recogntion accuracy of 95 percent(5).

2.7 Summary

In this chapter the operation, training, and testing procedures of several face recognition systems have been described. The performance of these systems are used as benchmarks by which to gauge the system developed in this thesis. Many research efforts have achieved various levels of success in developing the image collection, segmentation, feature extraction, and classification algorithms necessary to perform face recognition. A common thread in each effort described here is that each utilized holistic or global feature sets; however, the specific method employed by each varies greatly. Also, it is apparent that each of these systems has been tested under laboratory conditions which do not necessarily reflect a real world environment. The most common weakness was that test and training images were collected in a single sitting. In a practical application, training images can be collected at a single sitting if desired but test images must be collected throughout the operating lifetime of a face recognition system. In addition, constraints such as neutral background as well as constant distance, position and lighting should be eliminated. The algorithms should be integrated into a single, self-contained application capable of performing face recognition in an end-to-end process from image collection thru preprocessing and feature extraction to classification.

III. Methodology

3.1 General

The main objective of this thesis was to implement and test an end-to-end, face recognition system to determine the practicality of its use in real world environments such as an office. Three versions of the system were implemented (see figure 3.1). All three versions make use of a common front end to collect, segment and preprocess the images. This front end is the result of integrating the preprocessing algorithms developed by Suarez(17) with a real-time image collection and face segmentation algorithm developed by Gay(4). The difference in the three system versions concerns the feature set and classifier utilized by the given version. The first version uses the Karhunen Loéve Transform (KLT) feature set and a K-nearest neighbor (KNN) classifier. The second version maintains the KLT feature set but employs a backpropagation neural network classifier (BPNN)(8). Finally the third version uses the Discrete Cosine Transform (DCT) as the feature set and inputs coefficients into a KNN classifier. The software which implements these three versions is presented in APPENDIX A.

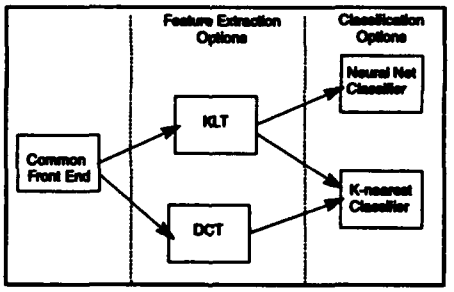


Figure 3.1. Three Versions of the Face Recognition System

3.2 Common Front End Processing

All versions of the software make use of a common set of front end processes. The function of these processes is to collect an image of an individual and prepare it for whatever feature extraction process is desired. The algorithms are the same regardless of which feature set or classifier is used or whether the system is training or recognizing. The block diagram for the common front end is shown in figure 3.2

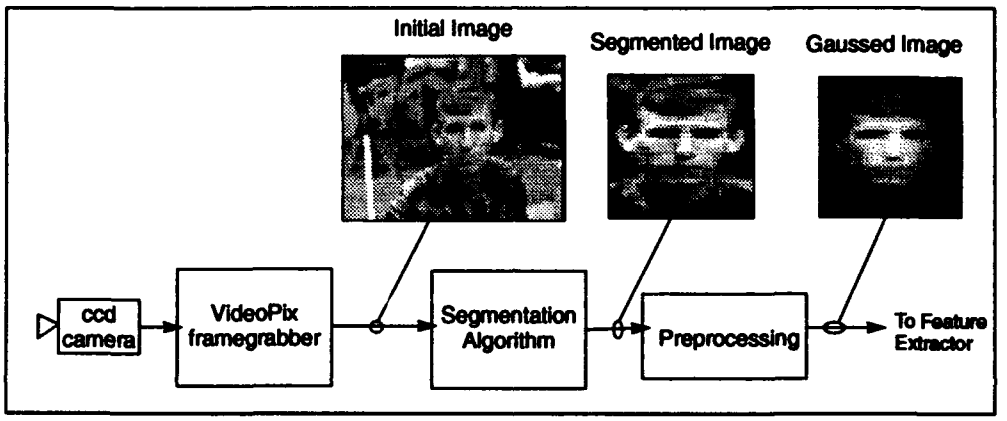


Figure 3.2. Common Front End System

3.2.1 Image Collection and Segmentation The segmentation algorithm developed by Gay in a collatoral thesis(4) includes the capability to grab images in real time using a CCD camera and a VideoPix image processing board (again refer to figure 3.2). The algorithm is based on motion detection using frame subtraction and slope analysis to determine the portion of the image which is most likely the head and segment that region from the 640 x 480 original image. The resulting image is a 32 x 32 pixel image. The algorithm is for the most part independent of background although as stated in chapter one, the assumption is that the only motion in front of the camera is the user. To an extent

the segmentation algorithm reduces sensitivity to scale that normally breaks down the KLT algorithm. This is because the entire area of motion is mapped into the same 32 x 32 pixel image no matter what distance the user is from the camera. Gay's thesis explains this side benefit in more detail.

3.2.2 Preprocessing as done by Suarez consists of two algorithms. The first procedure, CENTER.C, does a correlation between a reference image and each input image. The reference image can be any image where the subject is fairly well centerd in the picture. The offset of the peak correlation is used to shift the contents of the image such that the face in each picture is centered. After the image is centered it is multiplied by a gaussian window using GWIND.C. The logic here is to enhance the inner part of the picture (the facial features) while de-emphasizing the outer portion of the image (the background). The image is then again centered using CENTER.C; however the reference image is now a gaussianed version of the original correlation reference. Suarez's software has been modularized as C procedures and the interfaces redesigned to automate the original process. *CENTER.C was derived* from Suarez's C_LATE5.C. The modularized version of GWIND.C was not renamed.

3.3 Feature Extraction

As mentioned previously, two versions of the recognition system make use of a feature set based on KLT. The third version employs the DCT feature set. The software which performs these functions and the modifications made to the original versions are described in the following paragraphs.

3.3.1 Karhunen Loeve Transform The KLT software consists of two modules (see figure 3.3). KLT.C calculates the average face of the training set as well as the basis set of eigenfaces from which the KL coefficients are extracted. KLT.C is a modified version of Suarez's KL_TRANSFORM2.C. COEFFICIENTS.C calculates the coefficients which when multiplied by the eigenfaces in a linear

combination and then summed with the average face provide the least mean square error reconstruction of the original image. These coefficients are written to a data file along with the subject name associated with those coefficients. There are two versions of the coefficients software. COEFFICIENTS.C provides a data file of coefficients consistent with that required by the KNN classifier, while NET_COEFFICIENTS.C results in a data file formated for the BPNN classifier. Both versions of the coefficients routine were derived from Suarez's RECON.C algorithm.

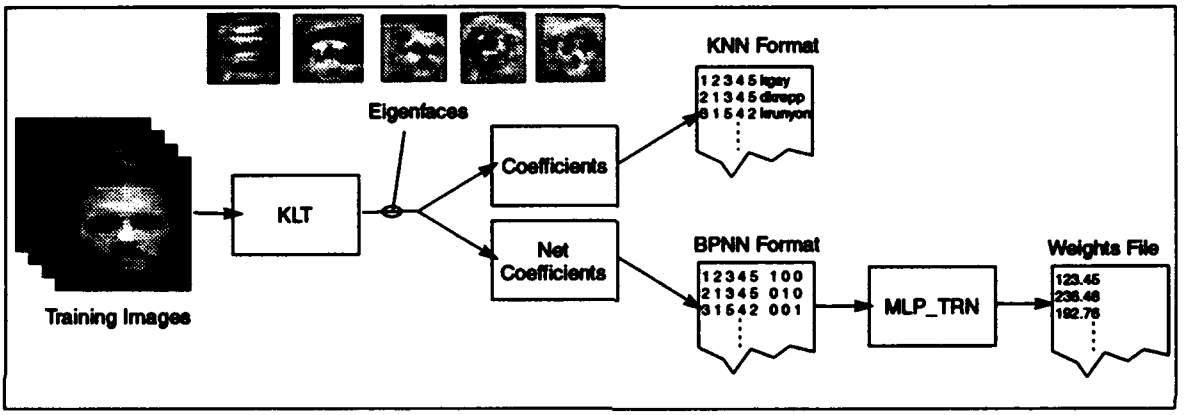


Figure 3.3. KLT Feature Extraction

3.3.2 Discrete Cosine Transform The DCT software consists of only one algorithm. The algorithm simply calculates a specified number of DCT coefficients and stores them in a data file along with the name of the subject associated with the image(see figure 3.4). The routine is a modified version of Goble's DCT.C algorithm and is called MDCT.C.

3.4 Classifiers

As mentioned before the face recognition system will be implemented in three versions. The first two will differ in the feature set utilized with the first being the KLT and the second the DCT.

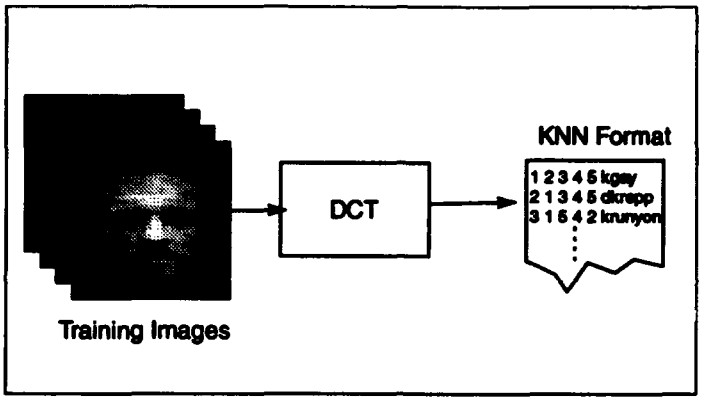


Figure 3.4. DCT Feature Extraction

Both will send coefficients to a K-nearest neighbor classifier to do recognition. A third version will again use the KLT feature set but the coefficients will be input to a backpropagation neural network for classification. The algorithms which implement the two classifiers are described below.

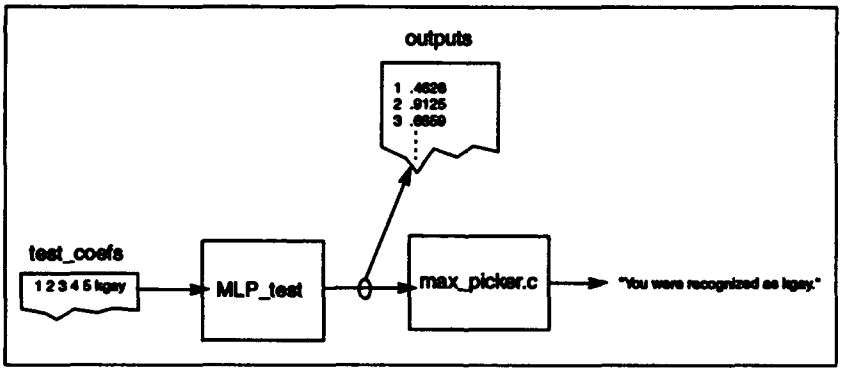


Figure 3.5. Back Propagation Neural Net Recognition Process

3.4.1 Backpropagation Neural Network The neural network algorithm used in this application was developed in an earlier AFIT dissertation(13). These algorithms have been modifed and tested in a collatoral thesis(8). The routine MLP.C is compiled in both a training configuration and a test configuration. MLP_TRN is compiled without the no train (NOTRN) flag set while MLP_TEST is

compiled with NOTRN set. The difference between the two is that the training configuration includes an iteration loop to update and learn the weights necessary to classifiy a given image in a training set. The testing configuration is compiled to read the weight file which was stored in the training procedure, pull in the coefficient file for the test image and output the name of the individual associated with the output node which responds.

3.4.2 K-nearest neighbor K_NEAREST.C is an implementation of a K-nearest neighbor classifier where K is chosen to be the number of training images used for each face in the training set. As shown in figure 3.6 the inputs to the classifier algorithm are the train_coefs file which was created in the training phase, and the test_coefs file generated from the input image by the recognition software. The resulting output is a print statement identifying the subject in the test image. The K-nearest neighbor uses a scoring technique in which the prototype nearest the test image receives a score of K while the second nearest neighbor receives a score of K-1. The most distant K neighbor receives a score of one. After the scoring of the neighbors is complete, all of the scores for a given name are summed together. The name associated with the highest score is selected as the identity to the test face. This procedure eliminates inaccuracies associated with a simple nearest neighbor which are caused by outlier training faces.

3.5 Training Software

The modular routines for feature extraction and classification, as well as the grab, segmentation, and preprocessing routines can be combined to generate various versions of training and testing programs where the feature extraction and/or classifier routine can be easily changed by calling a different module. In addition programs can be written to use previously stored images (without

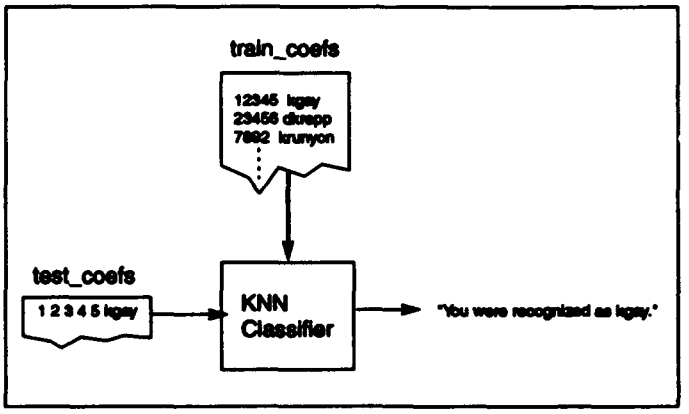


Figure 3.6. K-nearest neighbor Recognition Process

performing a grab) and those images can either be preprocessed or not, depending on the designer's needs. Three training programs have been assemble from these modules to implement the versions of the system outlined in this thesis. The descriptions of these programs are given below.

3.5.1 KLT KNN System TRAIN.C implements the KLT feature set and generates coefficients in a format compatible with the K-nearest neighbor classifier. The flow of the program is shown in figure 3.7. This program performs a user interactive training image collection, preprocesses the images, forms the eigenface basis set and extracts the desired number of coefficients from the training images. Those coefficients are then written to the TRAIN_COEFS file to be used by the recognition program. TRAIN.C is executed by entering the command "train" at the command line.

3.5.2 DCT KNN System TRAIN_DCT.C has much of the same functionality as TRAIN.C. The exception is that the resulting coefficients are calculated using the DCT rather than the KLT. The gray box in figure 3.7 shows the alternate flow followed by TRAIN_DCT.C. Like TRAIN.C this program performs a user interactive training image collection, preprocesses the images, and extracts the desired number of coefficients from the training images. Those coefficients are then written to the

TRAIN_COEFS file to be used by the recognition program. TRAIN_DCT.C is executed by entering the command "train_dct" at the command line.

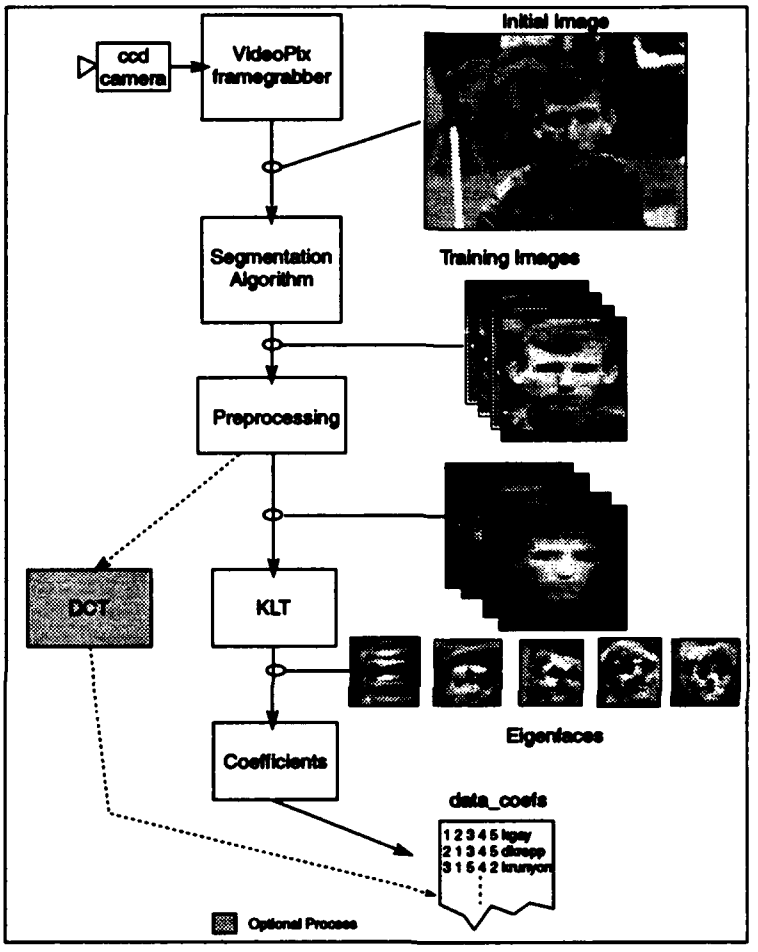


Figure 3.7. TRAIN.C/TRAIN_DCT.C Flow

3.5.3 KLT BPNN System The flow diagram of TRAIN_NET.C is given in figure 3.8. TRAIN_NET.C implements the KLT feature set and generates coefficients in a format compatible with the multi-layer perceptron classifier. This program performs a user interactive training image collection, preprocesses the images, forms the eigenface basis set and extracts the desired number of coefficients from the training images. Those coefficients are then used by MLP_TRN to calculate a set of weights to form

the decision surface which separates the training classes. These weights are stored in a weights file to be used by the recognition software. TRAIN_NET.C is executed by entering the command "train_net" at the command line.

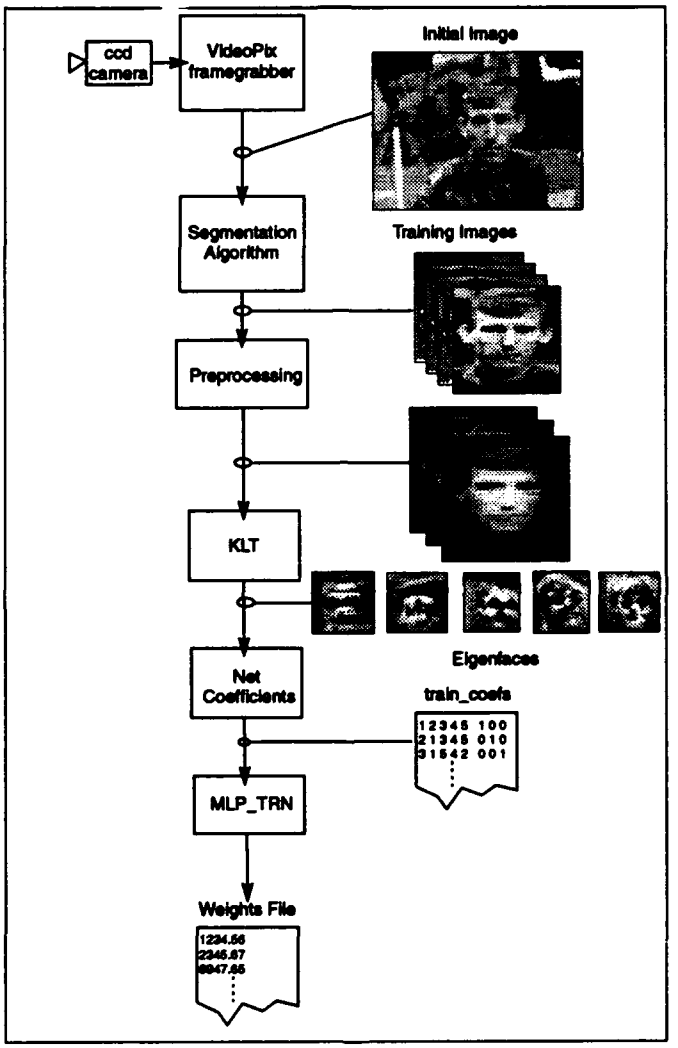


Figure 3.8. TRAIN_NET.C Flow

3.6 Recognition Software

For each of the three versions of training software, a corresponding version of recognition software has been developed. The names and descriptions of those routines are given below.

3.6.1 KLT KNN System VERIFY.C implements the recognition routine utilizing the KLT feature set and the k-nearest neighbor classifier (refer to figure 3.9). This routine performs a user interactive test image collection process, preprocesses the test image and extracts the KLT coefficients from that image. These coefficients are then sent to the k-nearest neighbor for classification. VERIFY.C is executed by entering the command "verify" at the command line.

3.6.2 DCT KNN System VERIFY_DCT.C again, makes use of most of the modules used in the KLT recognition software. The major difference is that the DCT coefficients are calculated for the test image instead of the KLT coefficients. The program performs a user interactive test image collection process, preprocesses the test image and extracts the DCT coefficients from that image. These coefficients are then sent to the k-nearest neighbor classifier for classification. VERIFY_DCT.C is executed by entering the command "verify_dct" at the command line.

3.6.3 KLT BPNN System VERIFY_NET.C implements the recognition routine utilizing the KLT feature set and the multi-layer perceptron classifier as shown in figure 3.10. This routine performs a user interactive test image collection process, preprocesses the test image and extracts the KLT coefficients from that image. These coefficients are then sent to the neural net classifier for classification. VERIFY_NET.C is executed by entering the command "verify_net" at the command line.

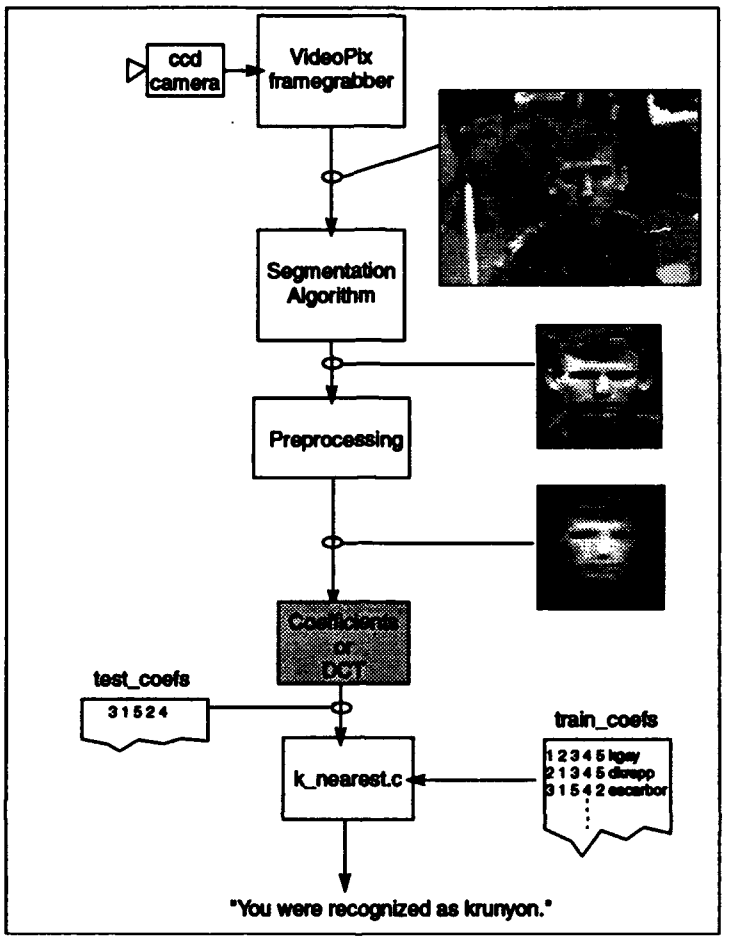


Figure 3.9. VERIFY.C Flow

3.7 Test Descriptions

In his thesis, Suarez evaluated his feature extraction software using a database of 55 users with four images of each user(17). With the addition of the real-time grab and segmentation algorithms, the constraints on the input images for this thesis can be relaxed to better emulate real world conditions. Thus a new and much less rigid training set has been collected to more fully test the robustness of the system.

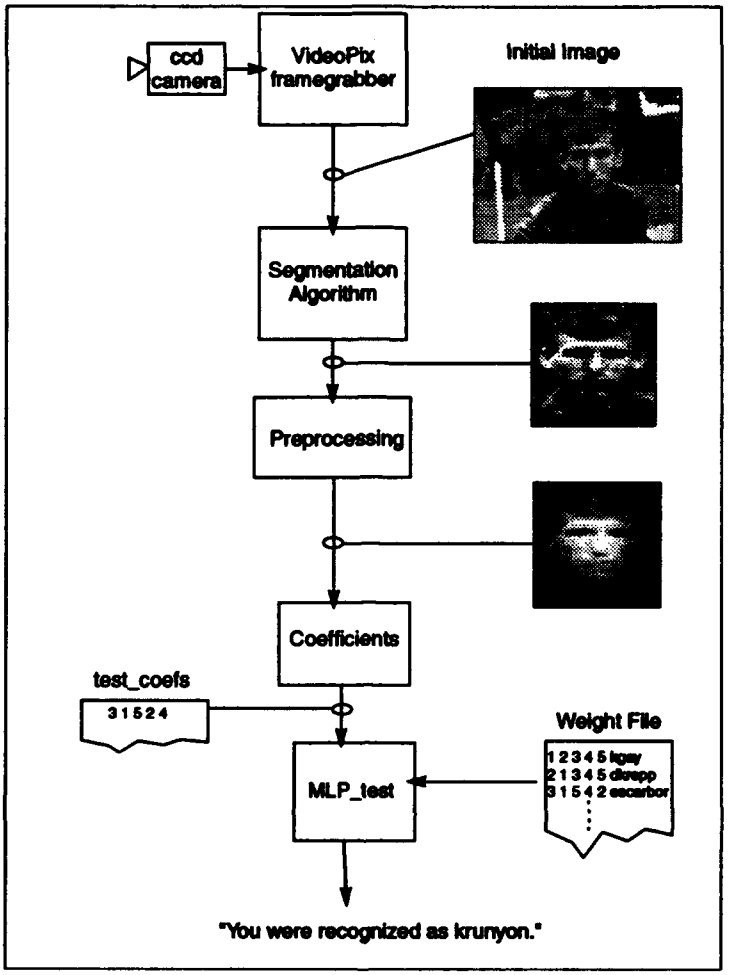


Figure 3.10. VERIFY_NET.C Flow

The KLT KNN and KLT BPNN systems were evaluated using three different test methods and two different databases. The first test gauged the accuracy of the systems for a moderate sized training set collected over a short period of time. The second test utilized a training set of only four people but provides detailed results of performance over a seven day period of time. The third test made use of the previous training sets and provides a very detailed evaluation of the systems performance over a large number of recognition attempts. The following sections provide the details of each test.

3.7.1 23 Subject - Two Day Test The data set consisted of two sets of images. The first set contained 115 images, 5 images for each of 23 users. The second training set also consists of 115 images, 5 images for each of the same 23 users. The difference between the two data sets is that they were taken on two different days. Same day recognition of the system was evaluated by first training the system on 4 images each of the 23 subjects collected on the first day. The fifth image of each subject was then input to the system as a test image. After that, the same procedure was accomplished for the images collected on the second day. The system was trained with four of the five pictures of each subject and tested with the fifth. The total number of correct recognitions for both days was then summed together and divided by the total number of tests to arrive at a standard same day recognition rate. The system was then trained on all 115 images from the first day and tested using all 115 images from the second day. Conversely, the system was trained using the 115 images from the second day and tested using the images from the first day. The total recognition accuracy for the two tests was again calculated. Finally, the system was trained using two images of each subject from the first day, and two images of each subject from the second day. The remaining fifth image for each subject for each of the two days (total of 46 images) was then supplied as the test set for the system.

3.7.2 30 Subject Manually Segmented - Two Day Test To test the effect of the segmentation algorithm on the overall recognition accuracy of the system, a data set which was collected in a collatoral thesis was used. The data set consisited of 300 total images of 30 different subjects. Each subject had 10 images in the data set. 5 of the 10 images were collected at an initial sitting, and the remaining 5 were collected at a later sitting. The subjects were positioned in front of a neutral background at a constant distance from the camera. The 23 Subject test procedure was performed using this data. The results of the two test were compared.

3.7.3 Four Subject - Seven Day Test The objective of the second test was to determine if the system improved when training images were taken over a period of time. To make this evaluation, a training set of four users was collected over a period of seven days with 7 images of each user taken everyday (four training images and three test images). The system was then trained using only the images of the first day. The one day system was then tested using each of the 21 test images collected over the seven day period. After recording the recognition score for the training set, the system was trained again using images from both the first and second day. The 21 test images were then run again and the score recorded. This additive training technique was continued, testing the 21 images each time, until the full seven day training set was utilized. Recognition accuracy for each training set was documented.

3.7.4 Long Term Recognition Test The objective of this test was to determine the recognition accuracy for an individual user over many recognition attempts. Several of the training sets from previous tests were used. Three tests were run using the KLT versions of the 23 Subject - Two Day Test data. The systems were trained using the same day, different day, and multiple day versions of the 23 subject training images. Each system was then trained using the best results from the Four Subject training set. Ten recognition attempts were made on both systems each day for ten days (a total of 100 recognition attempts for each system) with the same person attempting to be recognized each day.

3.8 Summary

Three versions of a face recognition application are developed in this thesis. The difference in the three versions are the variations of the feature extraction and classification algorithms used. The first version uses the Karhunen Loéve Transform to extract features and a K-nearest neighbor as

the classifier. The second version also makes use of the KLT but feeds the coefficients obtained to a backpropagation neural network classifier. Finally, version three uses the Discrete Cosine Transform to extract the features which are then input to a K-nearest neighbor classifier. The KLT based systems will be analyzed in a series of recognition tests to establish benchmark accuracies. The results of these tests are reported in the next chapter.

IV. Results

4.1 Introduction

This chapter describes the results of tests performed on the face recognition application developed in this thesis. Three separate versions were implemented in this thesis. All have the same image collection, segmentation, and preprocessing routines. They differ in the feature extraction algorithm and/or the classification scheme. The first version is based on a Karhunen Loéve Transform (KLT) feature extraction algorithm and makes use of a K-nearest neighbor classifier (KNN). The second version differs in that it employs a Backpropagation Neural Network (BPNN) classifier as opposed to the K-nearest neighbor. These two versions will be tested to document benchmark accuracies to compare against applications developed by other researchers. Finally, the third version which is based on the Discrete Cosine Transform (DCT) feature extraction algorithm and uses the K-nearest neighbor classifier was implemented to demonstrate the ability to easily make updates, changes, or replacements to any algorithm in the application.

Testing was accomplished using five image sets, each differing in the number of subjects and the number of sittings over which the images of a particular subject were collected. Various combinations of training and test images were drawn from these five data sets to explore the strengths and weaknesses of each version of the recognition application. All images were collected with the grab and segmentation routines developed for the recognition applications with the exception of one set of images which was manually segmented to evaluate the effect of automated segmentation on the overall accuracy. The five tests are denoted by the image set used for the test and are uniquely defined by the number of subjects and the time period over which the images were collected. The tests which made use of segmented data are:

- 23 Subject - Two Day Test
- Four Subject - Seven Day Test
- Single Subject Verification (Combination of Previous Image Sets)
- Single User - Long Term Test (Collected Over Several Weeks)

The 30 Subject - Two Day Test made use of images which were manually segmented. The details and results of each test are described in the following paragraphs.

4.2 23 User - Two Day Test

The purpose of this test was to determine the accuracy of the system for a fairly large number of subjects where the training and testing images were collected on different days. The image set used consisted of a total of 230 images, ten images for each of 23 subjects. The first five images of each subject were collected in an initial sitting. The last five images of each subject were collected a number of days later in a second sitting.

The feature set for the KNN system consisted of eight coefficients while the BPNN made use of 20 coefficients and 40 hidden layer nodes leading to 23 output nodes (one for each subject in the image set). Rather than use the same number of coefficients for each version of the application, the number of coefficients for which the algorithm seemed to perform best was used. This best number of coefficients was determined in an iterative trial and error process. Three versions of the test were run to evaluate the recognition accuracy of both the KLT/K-nearest neighbor(KNN) and KLT/neural network (BPNN) recognition applications.

Table 4.1. Three test results using image sets collected over two days. Results are given for test images collected on the SAME DAY as the training images, a DIFFERENT DAY than the training images, and for test and training images collected over MULTIPLE DAYS. Columns one and two are for 23 class data which was automatically segmented. Columns three and four show the results of comparable tests using 30 class data which was manually segmented. The KNN results are based on an eight coefficient feature set while the BPNN used twenty coefficients

	KNN	BPNN	Manual Segment KNN	Manual Segment BPNN
SAME DAY AVG	78	76%	90%	97%
DIFFERENT DAY AVG	29%	34%	40%	53%
MULTIPLE DAY RESULT	67%	74%	85%	95%

4.2.1 Same Day Test The purpose of this test was to obtain a recognition accuracy which could be compared to the tests performed by Suarez(17). Suarez's images were collected during a single sitting for each subject and had constraints such as neutral background, constant distance, etc., in order to appropriately test the KLT feature set. The aim of the current test is to determine the decrease in recognition accuracy due to loosening those constraints and allowing the segmentation algorithm to locate the subject in an uncontrolled background.

Both the KNN and BPNN applications were initially trained using four images for each of 23 subjects. A fifth image of each subject, which was not included in the training set, comprised the test set. Both the test and training set images for each subject were taken at the initial sitting. The 23 test images were input to both the KNN and BPNN system. Individual scores were then calculated and recorded for each system. Following that each system was retrained using four images of each subject taken at the second sitting. The test set consisted of the fifth image for each subject also taken at the second sitting. Both the BPNN and KNN were again tested on the second 23 image test set. The recognition scores for both systems were again determined and the score form the first and second days were averaged together to determine an overall same day recognition score. Table 4.1 shows that the

KNN achieved an overall same day score of 78 percent while the BPNN correctly identified 76 percent of the test images. These numbers are significantly lower than the 95% Suarez(17) achieved using his data set due to the constraints Suarez used in gathering his data (see Effect of Segmentation section).

4.2.2 Different Day Test The purpose of this test was to determine the potential decrease in recognition accuracy due to training on a set of images and then testing on images of the same subjects but collected at a later date. The training set consisted of four images of each subject collected on the first day. The test set was the five images of each subject collected on the second day. The recognition score for both systems were calculated and the training and test sets were then reversed such that the training set was made up of four images for each subject from the second day and the test set was the five images of each subject collected on the first day. The recognition accuracy was again determined and the results of the two tests were averaged into an overall score. For this version of the test, the KNN decreased to an overall score of 29 percent while the BPNN fell to 34 percent (see table 4.1). As expected the accuracies are markedly lower than those of the SAME DAY TEST. It appears that subtle changes in both the subject (hairstyle, facial features, clothing) and the environment (lighting, background bordering the user's face, distance to camera) have enough effect to cause errors to occur.

4.2.3 Multiple Day Training The purpose of this test was to determine if the decrease in accuracy due to training and testing on different days could be overcome by training on images from both days. In addition this test determines to an extent whether or not superposition applies to the Karhunen Loéve Transformation. The expectation was that if the training images were used from two separate days and only images collected on those two days were included in the test set, the resulting recognition accuracy should be close to the average recognition accuracy of the same day tests.

Both the KNN and BPNN systems were trained using four training images of each subject from each of the two collection days. The test set consisted of the remaining two images, each collected on a different day, of each subject. The KNN raised to an accuracy of 67 percent while the BPNN returned to 74 percent. While the BPNN was somewhat close to the original same day accuracy, the KNN result was low to be considered approximately equivalent to the previous score. Here it seems as though the BPNN can better generalize with the limited data available such that the effect of subtle changes in the environment and/or the user can be de-emphasized. Given a greater number of training images, both systems would tend toward their respective SAME DAY TEST accuracy.

4.3 Effect of Segmentation

The purpose of this test was to determine the change in overall accuracy of the system due to the segmentation algorithm. To perform this test, a data set was obtained from a collateral thesis effort(8). This data set consisted of ten images each of 30 different subjects. Like the original data set, five images of a given subject were taken during an initial sitting and the remaining five were collected at a later sitting. The difference was that this data set was collected in a controlled environment (neutral background, constant distance from camera, and careful positioning of the test subject) and segmentation was performed manually.

The Same Day, Different Day, and Multiple Training Day tests were repeated using the new images. The results as shown in columns three and four of Table 4.1 show that the segmentation algorithm decreases overall accuracy by approximately 20 percent. There are several factors which contribute to the lower auto-segmentation score. The segmentation algorithm is based on the user's motion envelope during the image collection period. If displacement is greater during one image than that of the next, the scale may be somewhat different. The KLT is very sensitive to scale(17) as well

as shift and rotation. Shift is not as much of a problem due to the correlation routine used to center the subject in the image. Sensitivity to scale could be reduced by gathering more training images.

4.4 Four Subject - Seven Day Test

The purpose of this test was to provide a more detailed analysis of the sensitivity of both the KNN and BPNN applications to images taken over time. While the 23 user test points out the detrimental effects of time on large systems, the four user test attempts to provide a solution to the problem albeit for a smaller number of users. The data set for this test was made up of 196 images. There were four individuals in the data base with 49 images each. Each subject made seven visits to the lab to have images collected. Four training and three test images were collected for each subject each day for a total of 28 training images and 21 test images for each of the four people.

The procedure for testing was an iterative process. First each system was trained on each person's four training images from day one and then tested on all 21 test images of each person. The system was then retrained using the training images from the second day in addition to the first. The accuracy of the system was again tested using all 21 test images for each subject. The system was then trained using three days of training images, tested, trained again, and so on until the training images for all seven days had been used.

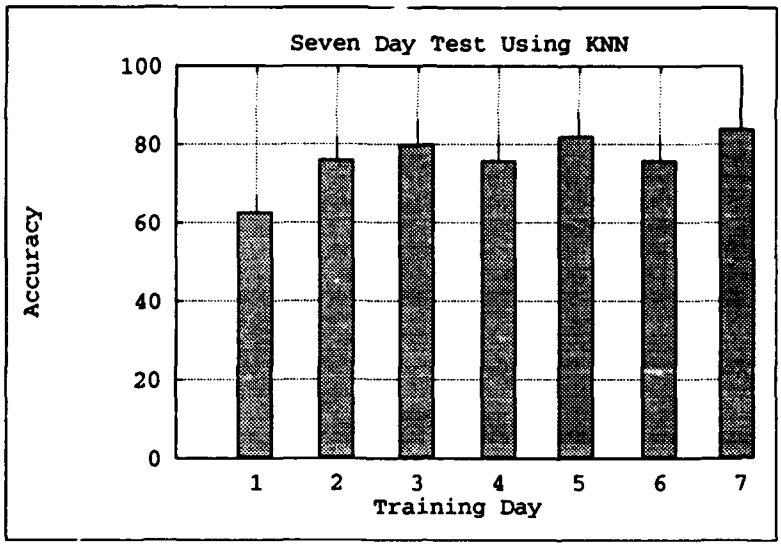


Figure 4.1. Recognition accuracies for four class data. Four KLT coefficients were used. The training and test sets were collected over seven days. The same test set is used for every training day and includes all test images from all seven days. The training set for each column consists of all training images collected up to and including the training day listed on the X-axis.

4.4.1 K-nearest neighbor The KNN system was trained using four KLT coefficients for each image. This number was determined based on the results of the 23 subject test and differs with the results of Suarez(17) who determined that the number of coefficients used should be roughly equal to one third the number of classes to be recognized. It appears that as the number of training images increase for each class, the number of coefficients must also increase in order to adequately separate them in KL space. Suarez always used four training images per class. No hard and fast equation involving the number of training images and the number of classes is as yet apparent. The results for each iteration of the KNN system are given in figure 4.1. Notice that the accuracies decreased on day four and six even though the training set contained more training images.

Table 4.2 shows the accuracy for a given test set at each training iteration. It was expected that the accuracy for each training set would fall off drastically once the test sets which had no corresponding

Table 4.2. Recognition accuracies for four class data using a K-nearest neighbor classifier. Four KLT coefficients were used. The seven test sets were collected on each of seven days. Each column shows the accuracy for a given day's test set as the system is iteratively trained on the images from an increasing number of training days.

	Train 1	Train 1,2	Train 1-3	Train 1-4	Train 1-5	Train 1-6	Train 1-7
# Prototypes	4	8	12	16	20	24	28
Test Set 1	92%	100%	92%	83%	100%	92%	92%
Test Set 2	58%	58%	67%	58%	75%	58%	75%
Test Set 3	67%	83%	92%	83%	100%	83%	92%
Test Set 4	67%	100%	92%	83%	100%	92%	100%
Test Set 5	67%	58%	75%	75%	75%	75%	92%
Test Set 6	50%	83%	83%	67%	67%	75%	83%
Test Set 7	67%	92%	100%	100%	100%	100%	100%
OVERALL	67%	82%	86%	82%	88%	82%	90%

training images in the training set had been reached; however, the accuracy for a given test set does not seem to correspond to whether it has images in the training set or not. These results show that differences in scale, shift, and rotation are more important than day to day differences in the subject and/or the image collection environment. Note also that the accuracies for all test sets in column one are much higher than the different day results from the 23 class KNN test althought with the exception of the number of classes, these tests are very much the same procedure. The reason the results are so much higher for the four class test is that the statistical chances of guessing correctly are much higher for four classes than for 23 classes. In addition there is less ambiguity or overlapping with four classes than with 23 classes, even though eight coefficients were used in the 23 class test and only four were used in the four class test.

4.4.2 Back Propagation Neural Network The network was configured with 20 input nodes, 40 hidden layer nodes and four output nodes. Again, this configuration was based on results from the previous 23 User Test. The results for each iteration of the BPNN system are given in figure 4.2. Here

the recognition accuracy continually increased as the training images from each day were included in the training set. This differs with figure 4.1 where the accuracy was sometimes lower after adding a new day of training images. This leads to the conclusion that the BPNN classifier generalizes more effectively on the available training images than does the KNN.

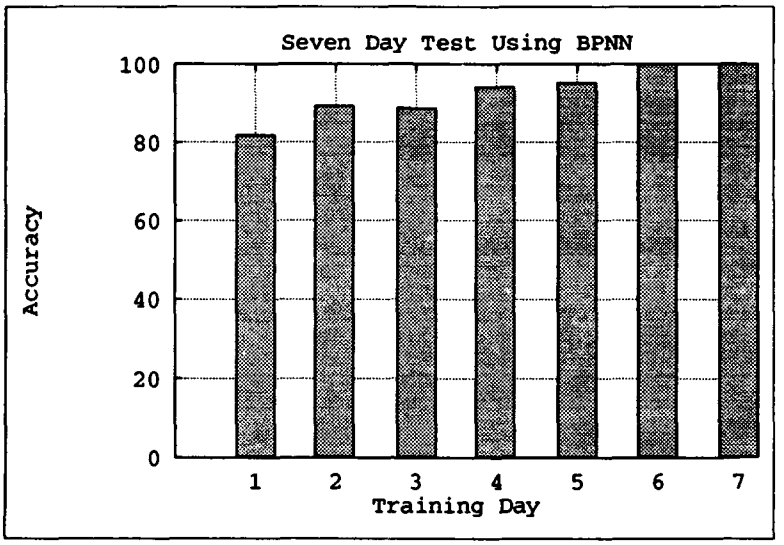


Figure 4.2. Recognition accuracies for four class data using a BPNN Classifier and 20 KL coefficients. The accuracy is four 21 test images taken 3 per day for 7 days. The system was first trained using data from day 1, all 21 test images were tested and the system trained again using day 1 and day 2 training data. This iterative process continued until the final system was trained on all 7 training sets. The columns represent the accuracies of the entire test set on each of these systems.

Table 4.3 shows the accuracy for a given test set at each training iteration. In general, recognition accuracy does increase for each test set with the addition of a new day of training images. Again this is due to the ability of the BPNN to generalize on the training images. Also note that test sets with outlier problems score low using initial training sets but improve to 100% accuracy after several training sets are added to the system (see test set 6).

Table 4.3. Recognition accuracies for four class data using a BPNN Classifier and 20 KL coefficients. The accuracy is four 21 test images taken 3 per day for 7 days. The system was first trained using data from day 1, all 21 test images were tested and the system trained again using day 1 and day 2 training data. This iterative process continued until the final system was trained on all 7 training sets. The columns represent the accuracies of the entire test set on each of these systems.

	Train 1	Train 1,2	Train 1-3	Train 1-4	Train 1-5	Train 1-6	Train 1-7
# Coefficients	4	8	12	16	20	24	28
Test Set 1	92%	100%	100%	92%	100%	100%	100%
Test Set 2	75%	75%	75%	92%	92%	100%	100%
Test Set 3	92%	92%	100%	92%	100%	100%	100%
Test Set 4	92%	92%	92%	100%	92%	100%	100%
Test Set 5	75%	92%	92%	100%	100%	100%	100%
Test Set 6	58%	83%	75%	83%	83%	100%	100%
Test Set 7	92%	92%	92%	100%	100%	100%	100%
OVERALL	82%	89%	89%	94%	95%	100%	100%

4.5 Accuracy versus K for the K-nearest neighbor

Having run several tests to this point using the K-nearest neighbor voting scheme as a classifier, the data was analyzed to determine the most effective value of K for the algorithm. Two plots were made, one using the results of the fourth day of the Four Subject - Seven Day Test, and the other the Long Term Recognition Test results using the training data of the 23 Subject - Different Day Test with the KNN classifier. Figure 4.3 shows that accuracy increases with K while figure 4.4 shows that accuracy decreases with K. While the results of theses tests seem to be contradictory, further examination reveals that these systems represent two very different feature spaces. The eight coefficient feature space of the 23 class problem is of much higher than that of the four coefficient four class problem. Consequently, a more complex set of classes can be separated in that space. However, there are very few prototypes to define the boundaries of these classes. The sparse number of prototypes assign too much importance to a given prototype which may or may not be an outlier from another class. The results need to be

averaged to arrive at a more correct solution. Recognition accuracy won't be too good regardless of the value of K due to poor boundary definition.

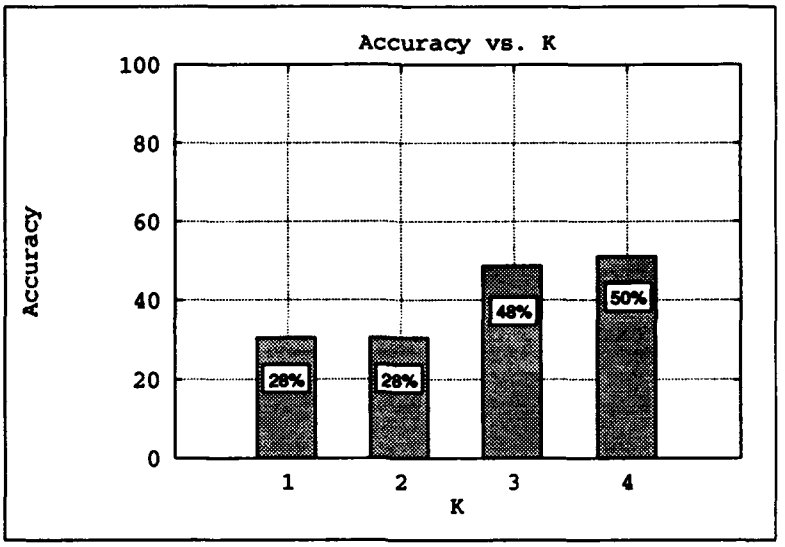


Figure 4.3. This graph shows the recognition accuracy obtained by testing 100 test images of the same subject collected over several months. The training set consisted of 23 classes with 8 prototypes collected four at a time on each of two days. Each column represents the accuracy associated with a particular K value for the K-nearest neighbor algorithm.

On the other hand the feature space defined by the second set of parameters is not quite as voluminous as the previous system; however, it contains only four classes. These classes are populated with a fairly large number of prototypes as compared to the rpior system. This dense population of prototypes provides fairly good results no matter what value of K is chosen. Still, the accuracy can be increased by reuding the number of prototypes used in the determination because prototypes on the fringe of a given class do not use prototypes on the fringe of some other class in the calculation. K should be kept to a value of 1 or 2.

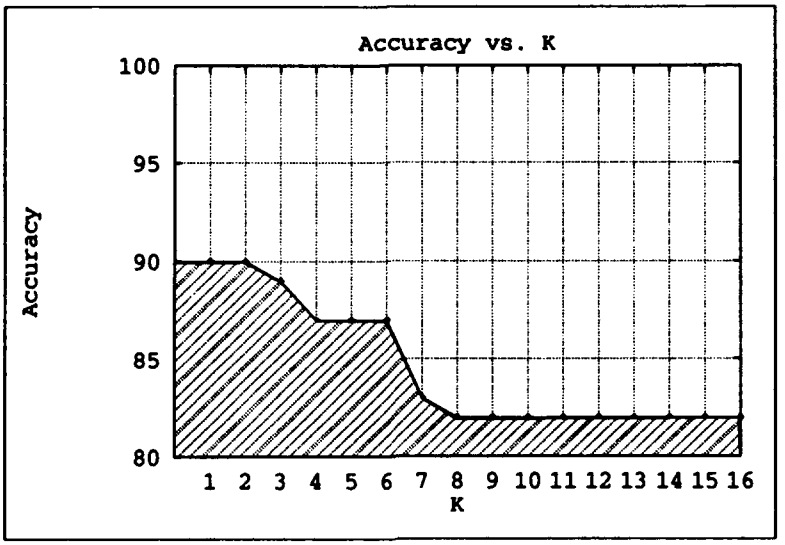


Figure 4.4. This graph shows the accuracy of testing 21 test images collected 3/day over 7 days on a system which was trained with 16 prototypes for each class. These prototypes were collected over 4 days. Each point on the curve represents the accuracy associated with a particular K value for the K-nearest neighbor algorithm.

4.6 Long Term Recognition Test

The purpose of this test was to simulate the login activity of a single user over a long period of time. The data set collected for this test is made up of a single user. There 100 images in the data set were collected five or ten at a time over a period of several weeks. All 100 images are used as a test set for seven of the previously trained systems. The configurations used from the 23 Subject - Two Day Test were both the KNN and BPNN version of the Different Day and Multiple Day Tests. The configurations tested in detail from the Four Subject - Seven Day Test were BPNN Day 7 and Day 6 as well as the KNN version of Day 7. Results for the test are given in table 4.4.

The difference between original results and the results of this test differs depending on what system is tested. Both versions of the different day test seemed to show improved recognition capability on the order of a 15

Table 4.4. All systems listed in column one were trained in previous tests. The accuracies obtained in the original test are shown in column two. Column three shows the accuracy of the given system when the original test set is replaced with a test set consisting of 100 images of a single subject collected over several weeks. The subject was a member of all previous training sets though none of the 100 images in the new test set were used in any training set.

	# Classes	# Coefficients	Original	Long Term
KNN Different Day	23	8	31%	50%
BPNN Different Day	23	20	37%	51%
KNN Multiple Day	23	8	67%	69%
BPNN Multiple Day	23	20	74%	55%
KNN Four - Day 7	4	4	90%	24%
BPNN Four - Day 7	4	20	100%	97%
BPNN Four - Day 6	4	20	100%	100%

4.7 Single Person Verification

The purpose of this test was to determine the verification accuracy for a BPNN verification system. The data set for this test was made up of two classes, a target class and a non-target class. The non-target training images consisted of 2 images of each of 22 subjects. Each of the two images was collected at separate sittings. The target training set consisted of 30 images of a single subject. Five images were collected at sitting one, five more at sitting two, and the last 20 images were collected four at a time over five sittings. The target images made up roughly forty percent of the training set with the non-target images making up the remaining sixty percent. The test set for the non-target class consisted of two images each for each of 22 subjects and again each image was collected at separate sittings. The target test set consisted of 10 images of the single subject. The first nine images were collected three at a time over three sittings and the tenth image was collected at a fourth sitting. All test images had corresponding training images (the training images came from the same sittings as the test images) but not all training images had corresponding test images.

The network was configured to accept twenty KL coefficients at the input layer. These inputs feed forty hidden layer nodes and of course two output nodes signified the target and non-target classes. The network was trained to a 100 percent accuracy on the training set in 600 iterations. The false accept rate for the network was two percent, while the false rejection rate was forty percent.

V. Conclusions

5.1 Introduction

This chapter states the conclusions formed by the results of the tests performed in chapter Four. The chapter begins with the conclusions drawn from the 23 Subject - Two Day Tests. Next the results of the 30 Subject, Manually Segmented - Two Day Test will be discussed. Insights provided by the Four Subject - Seven Day Test are then be provided. Then a short discussion on the accuracy of the K-nearest neighbor classifier versus the value of K will be given. Conclusions drawn from the Long Term Recognition Test will then be shown, and finally, the Single Person Verification Test is discussed.

5.2 23 Subject - Two Day Test

This test addresses the problems associated with training a system to recognize based on images taken at a specific instance in time and then expecting that training set to generalize to images taken at any time later. This multiple day problem causes large decreases in recognition accuracy. The lose in accuracy is the result of the subtle differences in both the image collection environment (such as luminance, distance to camera, shift, position, and scale) and the subject (such as hair shape, beard growth, facial appearance at different times of day) when viewed over a period of time. Little attention has been given to this problem, certainly none of the systems described in the literature search of chapter Two dealt with multiple day recognition. The two day test doucmented the fact that both the KNN and the BPNN fell off significantly. The KNN score fell from 78% to 29% while the BPNN system went from 76% to 30%. This problem can be overcome to a large extent by training over multiple days. The effect of training on multiple days is that potential variations due to time can be accounted for in the training set allowing the classifier to generalize over the differences. Using multiple day training

the KNN improved to 67% while the BPNN increased to 74%. The BPNN and KNN seemed to be approximately equal in all tests performed in this section although the neural net classifier seemed to regain more accuracy when trained over multiple days than did the K-nearest neighbor.

5.3 30 Subject Manually Segmented - Two Day Test

These tests indicated that the decrease in recognition accuracy due to the current automated segmentation algorithm is very close to 20% regardless of the training set, test set, or classifier type. It did appear that the neural net classifier substantially outperforms the k-nearest neighbor when using the manually segmented data. The smallest difference was the KNN score 90% for the SAME DAY TEST as compared to the BPNN score of 97%. The dropoff in recognition accuracy when training and testing on different days was still very apparent. The KNN score dropped to 40% while the BPNN fell to 53%. Both returned to their approximate SAME DAY values when trained over multiple days. In general, the lose in recognition accuracy is not worth the ability to segment using the current routine.

5.4 Four Subject - Seven Day Test

The recognition accuracy for both the KNN and BPNN systems increased as the number of days over which the systems were trained was increased. However, due to suspected outliers in some of the test sets, the recognition score for the later test sets were as good or better than the scores of the earlier tests sets even though the system had been trained on only one or two days of training images. For instance, at training day 2 in table 4.2, test set 7 is at 92 percent while test set 2 is still only 58 percent. The same situation occurs in the BPNN table. Notice that test set two generally fell below the score of later test sets even after the training set from day two had been included. Overall, the neural

net classifier performed substantially better for this particular series of tests than did the K-nearest neighbor.

5.5 Accuracy Versus K for the K-nearest neighbor

The two tests determined to evaluate this relationship seem to contradict each other. The result from the Long Term Recognition data for the 23 Subject Test would indicate that K should be as large as possible. The result from the Four Subject Test indicates that the K-nearest neighbor should be changed to a simple nearest neighbor for the optimal accuracy. To resolve this contradiction, the feature space of the system under test must be examined. For the Long Term Recognition Accuracy Test, the system is trained with the following parameters:

- 23 classes
- 8 KL coefficients
- 4 prototypes for each subject
- 92 training images total

The Four Subject Test is trained with the following parameters:

- 4 classes
- 4 KL coefficients
- 16 prototypes for each subject
- 64 training images total

These systems represent two very different feature spaces. The first set of parameters constructs a higher dimension feature space which is then expected to support a more complex set of classes. However, there are very few prototypes to define the boundaries of these classes. The sparse number of prototypes assign too much importance to a given prototype which may or may not be an outlier from another class. The results need to be averaged to arrive at a more correct solution. Recognition accuracy won't be too good regardless of the value of K.

On the other hand the feature space defined by the second set of parameters is not quite as voluminous as the previous system; however, it contains only four classes. These classes are populated with a fairly large number of prototypes as compared to the prior system. This dense population of prototypes provides fairly good results (>80 percent) no matter what value of K is chosen. But the accuracy can be increased by reducing the number of prototypes used in the determination because prototypes on the fringe of a given class do not use prototypes on the fringe of some other class in the calculation.

5.6 Single Person Verification

Using a training set with a ratio of 60 percent non-target images to 40 percent target images provides a verification system which is biased toward rejecting a given subject regardless if the subject is of target or non-target class. The false reject rate of 40 percent is probably higher than the normal user would tolerate. The false acceptance rate of 2 percent is good depending on the level of security the system must provide.

5.7 Long Term Recognition Accuarcy

Some configurations of the system seemed to improve when tested with the Long Term Accuracy test set. These systems include both versions of the Different Day Test and the KNN version of the Multiple Day Test. The systems which showed a decrease in accuracy where the BPNN version of the Multiple Day and the KNN version of day 7 of the Four Subject Test. The BPNN day 6 and day 7 Four Subject training sets did extremely well for all images tested on them. The single person verification configuration was also tested using the Long Term Accuracy test set and was found to provide a 41 percent false reject rate which was consistent with the score from the previous test set.

5.8 Comparison to Other Systems

Table 5.1 and 5.2 compare the performance of systems discussed in Chapter Two to the systems developed in this thesis for both multiple recognition and single person verification. The WISARD system provides the best performance of all same day tests; however, the system is expensive, very complex and requires significant computational resources. In addition the WISARD, as well as the other systems listed, are tested with test images collected at the same sitting as the training images. It is logical to think that the accuracy for those systems would significantly decrease for images collected over time. Finally, with the exception of the WISARD system, none of the recognition systems provides any method of real-time image collection or automated segmentation (WISARD has no segmentation but depends on huge number of prototypes).

	# Classes	# Sittings	# Protos	Accuracy
WISARD	16	1	200 - 400	100%
UCSD	20	1	8	99%
Suarez	55	1	4	95%
Goble	55	1	4	95%
AFRM	50	1	4	73%
KNN System	23	1	4	78%
KNN System	23	2	8	29%
BPNN System	23	1	4	76%
BPNN System	23	2	4	34%
KNN System	4	7	28	90%
BPNN System	4	7	28	100%

Table 5.1. Comparison of all Multiple Recognition Systems Discussed

	# Sittings	# Protos	False Reject	False Accept
WISARD	1	200 - 400	0%	0%
Los Alamos	1	5	8.7%	0.02%
Suarez	1	16	8%	8%
Goble	1	16	2%	3.5%
Current System	6	46 Non-target/30 Target	40%	2%

Table 5.2. Comparison of all Single Person Verification Systems Discussed

6.1 Makefile

```
#### Face Recognition Makefile
#### 2 Nov 92
HOST=grimm
HOME=/data5/krunyon
BIN = $ (HOME)/bin/$(HOST)
#BIN=./
BASEDIR=/usr/1.0-VFC
VFCLIB_DIR=$(BASEDIR)/vfc_lib
VFCSYS_DIR=$(BASEDIR)/sys
GRAB_ROUTINES=z_set_vfc_hw.o z_grab_gra.o z_store_image.o
MOTION_ROUTINES=$(GRAB_ROUTINES) z_find_diff.o z_median.o z_motion.o
SEG_ROUTINES=$(MOTION_ROUTINES) z_outline.o z_seg_regions.o z_reduce.o \
z_segment.o
train : train.c seg_grab.o $(SEG_ROUTINES) display.o center.o \
gwind.o rescale.o nrutil.o fourn.o klt.o eigsrt.o jacobi.o \
coefficients.o
cc -g -I$(VFCSYS_DIR) -I$(VFCLIB_DIR) -o train train.c \
seg_grab.o \
$(SEG_ROUTINES) $(VFCLIB_DIR)/libvfc.a \
display.o \
center.o \
gwind.o \
rescale.o \
nrutil.o \
fourn.o \
klt.o eigsrt.o jacobi.o \
coefficients.o \
-lm
verify : verify.c grab.o seg_grab.o $(SEG_ROUTINES) display.o center.o \
  gwind.o rescale.o coefficients.o k_nearest.o nrutil.o fourn.o
cc -g -I$(VFCSYS_DIR) -I$(VFCLIB_DIR) -o verify \
verify.c \
grab.o \
seg_grab.o \
$(SEG_ROUTINES) $(VFCLIB_DIR)/libvfc.a \
display.o \
center.o \
gwind.o \
rescale.o \
coefficients.o \
k_nearest.o \
nrutil.o \
fourn.o\
-lm
train_dct : train_dct.c seg_grab.o $(SEG_ROUTINES) display.o center.o \
gwind.o rescale.o nrutil.o fourn.o mdct.o
cc -g -I$(VFCSYS_DIR) -I$(VFCLIB_DIR) -o train_dct train_dct.c \
seg_grab.o \
$(SEG_ROUTINES) $(VFCLIB_DIR)/libvfc.a \
display.o \
center.o \
gwind.o \
rescale.o \
nrutil.o \
fourn.o \
mdct.o \
-lm
verify_dct : verify_dct.c grab.o seg_grab.o $(SEG_ROUTINES) display.o \
center.o gwind.o rescale.o mdct.o k_nearest.o nrutil.o fourn.o
cc -g -I$(VFCSYS_DIR) -I$(VFCLIB_DIR) -o verify_dct \
```

```
verify_dct.c \
grab.o \
seg_grab.o \
$(SEG_ROUTINES) $(VFCLIB_DIR)/libvfc.a \
display.o \
center.o \
gwind.o \
rescale.o \
mdct.o \
k_nearest.o \
nrutil.o \
fourn.o\
-lm
train_net : train_net.c seg_grab.o $(SEG_ROUTINES) display.o center.o \
gwind.o rescale.o nrutil.o fourn.o klt.o eigsrt.o jacobi.o \
net_coefficients.o
cc -g -I$(VFCSYS_DIR) -I$(VFCLIB_DIR) -o train_net train_net.c \
seg_grab.o \
$(SEG_ROUTINES) $(VFCLIB_DIR)/libvfc.a \
display.o \
center.o \
gwind.o \
rescale.o \
nrutil.o \
fourn.o \
klt.o eigsrt.o jacobi.o \
net_coefficients.o \
-lm
verify_net : verify_net.c grab.o seg_grab.o $(SEG_ROUTINES) display.o center.o \
  gwind.o rescale.o net_coefficients.o nrutil.o fourn.o
cc -g -I$(VFCSYS_DIR) -I$(VFCLIB_DIR) -o verify_net \
verify_net.c \
grab.o \
seg_grab.o \
$(SEG_ROUTINES) $(VFCLIB_DIR)/libvfc.a \
display.o \
center.o \
gwind.o \
rescale.o \
net_coefficients.o \
nrutil.o \
fourn.o\
-lm
retrain : retrain.c  klt.o coefficients.o nrutil.o fourn.o eigsrt.o jacobi.o
cc -g -o retrain retrain.c klt.o coefficients.o \
nrutil.o fourn.o eigsrt.o jacobi.o -lm

seg_grab.o : seg_grab.c $(SEG_ROUTINES)
cc -g -I$(VFCSYS_DIR) -I$(VFCLIB_DIR) -c seg_grab.c \
$(SEG_ROUTINES) \
$(VFCLIB_DIR)/libvfc.a
center.o : center.c
 cc -g -c center.c
display.o : display.c
cc -g -c display.c
gwind.o : gwind.c
cc -g -c gwind.c
coefficients.o : coefficients.c
cc -g -c coefficients.c
net_coefficients.o : net_coefficients.c
cc -g -c net_coefficients.c
fourn.o : fourn.c
cc -g -c fourn.c
nrutil.o : nrutil.c
cc -g -c nrutil.c
rescale.o : rescale.c
cc -g -c rescale.c
```

```
klt.o :  klt.c
cc -g -c klt.c
mdct.o : mdct.c
cc -g -c mdct.c
jacobi.o : jacobi.c
cc -g -c jacobi.c
eigsrt.o : eigsrt.c
cc -g -c eigsrt.c
k_nearest.o : k_nearest.c
cc -g -c k_nearest.c
add_usr : add_usr.c seg_grab.o $(SEG_ROUTINES) display.o center.o \
gwind.o rescale.o nrutil.o fourn.o klt.o eigsrt.o jacobi.o \
coefficients.o
cc -g -I$(VFCSYS_DIR) -I$(VFCLIB_DIR) -o add_usr add_usr.c \
 seg_grab.o \
$(SEG_ROUTINES) $(VFCLIB_DIR)/libvfc.a \
display.o \
center.o \
gwind.o \
rescale.o \
nrutil.o \
fourn.o \
klt.o eigsrt.o jacobi.o \
coefficients.o \
-lm
#Grab Modules
z_set_vfc_hw.o : z_set_vfc_hw.c
cc -g -I$(VFCSYS_DIR) -I$(VFCLIB_DIR) -c z_set_vfc_hw.c
z_grab_gra.o : z_grab_gra.c
cc -g -I$(VFCSYS_DIR) -I$(VFCLIB_DIR) -c z_grab_gra.c
z_store_image.o : z_store_image.c
cc -g -I$(VFCSYS_DIR) -I$(VFCLIB_DIR) -c z_store_image.c
z_reduce.o : z_reduce.c
cc -g -I$(VFCSYS_DIR) -I$(VFCLIB_DIR) -c z_reduce.c
#Motion Modules
z_find_diff.o : z_find_diff.c
cc -g -c -I$(VFCSYS_DIR) -I$(VFCLIB_DIR) z_find_diff.c
z_median.o  : z_median.c
cc -g -I$(VFCSYS_DIR) -I$(VFCLIB_DIR) -c z_median.c
z_motion.o : z_motion.c
cc -g -I$(VFCSYS_DIR) -I$(VFCLIB_DIR) -c z_motion.c
#Segmentation Modules
z_outline.o : z_outline.c
cc -g -I$(VFCSYS_DIR) -I$(VFCLIB_DIR) -c z_outline.c
z_seg_regions.o : z_seg_regions.c
cc -g -I$(VFCSYS_DIR) -I$(VFCLIB_DIR) -c z_seg_regions.c \
$(VFCLIB_DIR)/libvfc.a
z_segment.o : z_segment.c
cc -g -I$(VFCSYS_DIR) -I$(VFCLIB_DIR) -c z_segment.c
grab_one.o : grab_one.c $(GRAB_ROUTINES)
cc -g -I$(VFCSYS_DIR) -I$(VFCLIB_DIR) -c grab_one.c \
$(GRAB_ROUTINES) \
$(VFCLIB_DIR)/libvfc.a
```

6.2 train.c

```
/*************************************************************
*  train.c
*
*  This program is used to train the KLT\KNN system.
*  The grab routine is called first to collect the training images.  After
*  all images of a particular user have been collected, each of the images
*  are preprocessed (centered and gaussian windowed).
```

```
*  The preprocessed images are then used by kl_transform to
*  create an average face and a user determined number of eigen faces.
*  The coefficients module is then called to extract the kl coefficients
*  from the training images.  These coefficients, which are the end result
*  of the training process are stored in train_coefs.  All training images
*  are stored in a folder called training_images for possible use in
*  retraining the system at a later time.
*
*  Written by:  Ken Runyon
*
*  Date:      21 Jul 92 - 31 Aug 92
*
*
*************************************************************/

#include <stdio.h>
#include <string.h>
#include "vfc_lib.h"
#include "globals.h"

int   i,
   finished,
   done,
   quit,
   num_protos,
   user_coefs,
   num_coefs,
   num_train,
   file_ptr[2];

FILE   *fparam,
   *flist,
   *fweights;

char   command[80],
   u_name[8],
   nu_name[8],
   filename[20],
   waste[2],
   another[4];

 main()
{

/********* Make a Training Folder to Hold the Prototypes *******/

system("mkdir training_images");

/*************** Open the Training List File ***************/

if ((flist=fopen("train_list","w+"))== NULL)
   printf("I can't open the train list.\n");

/*************** Open the Training Parameters File ***************/

fparam=fopen("train_params","w");

/********* Prompt User for Number of Prototypes *******/
done = 0;
while (!done){
  printf("\nEnter the number of prototypes to be used for each user <1-20>: ");
  scanf("%d",&num_protos);
  gets(waste);
  printf("\n");
  if ((num_protos ≤ 20) && (num_protos ≥ 1))
   done = 1;
  else
   printf("\nYou need to do at least 1 and at most 20.\n");
}
/****************** Enter Users Until You're Done *************/

while (!finished){
done = 0;
quit = 0;
```

```
/********* Prompt User for User Name ******/

 printf("\nEnter the person's username <8 letters>: ");
 gets(u_name);
 printf("\n");
 while(!done){
   printf("\nThe name you entered was : %s\n",u_name);
   printf("You can either re-enter a name or press return to continue.\n");
   gets(nu_name);
   if (nu_name[0] == '\0')
     done = 1;
   else {
     strcpy(u_name,nu_name);
     nu_name[0] = '\0';
   }
 }

/********** grab training images of the user **************/

file_ptr[0]=ftell(flist);
 for (i=1; i<= num_protos; i++){
   num_train = num_train++;
   sprintf(filename,"%s%d",u_name,i);
   seg_grab(filename);
   sprintf(filename,"%s%d%s",u_name,i,".gra");
   fprintf(flist,"%s\n",filename);
 }
 file_ptr[1]=ftell(flist);
 fseek(flist,(file_ptr[0]-file_ptr[1]),1);

/********* Preprocess the training images *****/

for (i=1; i<=num_protos; i++){
   fscanf(flist,"%s\n",filename);
   center(SM_WIDTH,"correlate.ref",filename);
   gwind(SM_WIDTH,filename);
   center(SM_WIDTH,"wind.ref",filename);
   sprintf(filename,"%s%d",u_name,i);
   display(SM_WIDTH,filename,num_protos);
   sprintf(filename,"%s%d%s",u_name,i,".gra");
 }

 printf("\nDo Another User? <y or n>: ");
 gets(another);
while (!quit){
 if ((another[0] == 'n') || (another[0] == 'N')){
   finished = 1;
   quit = 1;
 }
 else if ((another[0] == 'y') || (another[0] == 'Y')){
   finished = 0;
   quit = 1;
   }
 else {
     printf("\n\nHit y if you want to enter another user.\nHit n if you're done entering users.  :
");
   gets(another);
   printf("\n");
 }
}
}
/********** Decide how many eigenfaces you need **********/

done = 0;
while (!done){
  printf("\nEnter the number eigenfaces you want to train on <%d>: ",
 num_train/num_protos/3);
  scanf("%d",&user_coefs);
  gets(waste);
  printf("\n");
  if ((user_coefs > 0) && (user_coefs <= num_train)){
   num_coefs = user_coefs;
```

```
  done = 1;
  }
  else if (user_coefs == 13){
   num_coefs = num_train/num_protos/3;
   done = 1;
   }
  else
   printf("\nYou need to do at least 1 and at most %d\n",num_train);
}

fprintf(fparam,"%d\n%d\n%d\n%d\n",SM_WIDTH,num_coefs,num_train, num_protos);
fclose(flist);
fclose(fparam);

kl_transform("train_list", SM_WIDTH, num_train);

/********** Make Coefficients for All Training Images *********/

if ((flist = fopen("train_list", "r")) == NULL)
  printf("\nCollect can't open the training list.\n");

fweights = fopen("train_coefs","w");

for (i=1; i<=num_train; i++){
  fscanf(flist,"%s\n",filename);
  sprintf(command,"%s%s%s","mv ",filename, " training_images");
  coefficients(SM_WIDTH, num_coefs, filename, fweights);
  system(command);
}
fclose(flist);
fclose(fweights);
system("rm *.rle");
system("rm *.rec");
system("rm *.red");
printf("\nTRAINING IS COMPLETE\n");
}  /*END TRAIN*/
```

6.3 train_net.c

```
/*************************************************************
*  train_net.c
*
*  This program is used to train a system based on KLT feature
*  extraction and a neural net classifier.  The grab routine is first
*  called to collect the training images.  After all images of a
*  particular user have been collected, each of the images are
*  preprocessed (centered and gaussian windowed).
*  The preprocessed images are then used by kl_transform to
*  create an average face and a user determined number of eigen faces.
*  The coefficients module is then called to extract the kl coefficients
*  from the training images.  These coefficients are stored in a data
*  file called klt.dat to be used by the neural network training algorithm.
*  The neural network algorithm creates a weight file which will be used
*  in the recognition phase.  The outputs of this code are 1) the klt.dat
*  file, 2) the setup file for the network, and 3) the weight file created
*  by the network.  All training images
*  are stored in a folder called training_images for possible use in
*  retraining the system at a later date.
*
*  Written by:  Ken Runyon
*
*  Date:     25 Sep 92
*
*
*************************************************************/

#include <stdio.h>
#include <string.h>
#include "vfc_lib.h"
#include "globals.h"
#define  NUM_LAYRS  2
#define  WT_SED     1918940490
```

```
#define PART_SED   1191645590
#define  RNDM_SED   123456789
#define MAX_ITS      500
#define  OUT_INT      100
#define  ETA_IN      0.15
#define  ETA_OUT      0.3
#define ETA_1_2      0.0
#define  ALPHA       0.5
#define  BAT_SZ      1
#define  TRAIN_PCT   1.0
#define  NORM        1

int   i,
   finished,
   done,
   quit,
   month,
   day,
   select,
   num_protos,
   user_coefs,
   num_coefs,
   num_train;

   file_ptr[2];

FILE   *fparam,
   *flist,
   *fweights,
   *fset,
   *ftable;

char   wt_file[10],
   dat_file[10],
   command[80],
   image_folder[20],
   u_name[8],
   nu_name[8],
   filename[20],
   waste[2],
   another[4],
   num_class,
   hid_nodes,
   hid_nodes2;

 main()

{

/********* Make a Training Folder to Hold the Prototypes *******/

system("mkdir training_images");

/*************** Open the Training List File ***************/

if ((flist=fopen("train_list","w+"))==NULL)
   printf("I can't open the train list.\n");

/*************** Open the Training Parameters File ***************/

fparam=fopen("train_params","w");

/********* Prompt User for Number of Prototypes *******/
done = 0;
while (!done){
   printf("\nEnter the number of prototypes to be used for each user <1-20>: ");
   scanf("%d",&num_protos);
   gets(waste);
   printf("\n");
   if ((num_protos ≤ 20) && (num_protos ≥ 1))
   done = 1;
   else
   printf("\nYou need to do at least 1 and at most 20.\n");
}
/****************** Enter Users Until You're Done *************/
```

```
while (!finished){
done = 0;
quit = 0;

/******** Prompt User for User Name ******/

 printf("\nEnter the person's username <8 letters>: ");
 gets(u_name);
 printf("\n");
 while(!done){
   printf("\nThe name you entered was : %s\n",u_name);
   printf("You can either re-enter a name or press return to continue.\n");
   gets(nu_name);
   if (nu_name[0] == '\0')
     done = 1;
   else {
     strcpy(u_name,nu_name);
     nu_name[0] = '\0';
   }
 }
num_class++;

/********** grab training images of the user **********/

file_ptr[0]=ftell(flist);
 for (i=1; i<= num_protos; i++){
   num_train = num_train++;
   sprintf(filename,"%s%d",u_name,i);
   seg_grab(filename);
   sprintf(filename,"%s%d%s",u_name,i,".gra");
   fprintf(flist,"%s\n",filename);
 }
 file_ptr[1]=ftell(flist);
 fseek(flist,(file_ptr[0]-file_ptr[1]),1);

/******** Preprocess the training images *****/

for (i=1; i<=num_protos; i++){
   fscanf(flist,"%s\n",filename);
   center(SM_WIDTH,"correlate.ref",filename);

   gwind(SM_WIDTH,filename);

   center(SM_WIDTH,"wind.ref",filename);
   sprintf(filename,"%s%d",u_name,i);
   display(SM_WIDTH,filename,num_protos);
   sprintf(filename,"%s%d%s",u_name,i,".gra");
 }

 printf("\nDo Another User? <y or n>: \n");
 gets(another);
while (!quit){
 if ((another[0] == 'n') || (another[0] == 'N')){
   finished = 1;
   quit = 1;
 }
 else if ((another[0] == 'y') || (another[0] == 'Y')){
   finished = 0;
   quit = 1;
   }
 else {
     printf("\n\nHit y if you want to enter another user.\nHit n if you're done entering users.  :
");
   gets(another);
   printf("\n");
 }
}
}
/********** Decide how many eigenfaces you need **********/
done = 0;
while (!done){
```

```
  printf("\nEnter the number eigenfaces you want to train on <%d>: ",num_train/3);
  scanf("%d",&user_coefs);
  gets(waste);
  printf("\n");
  if ((user_coefs > 0) && (user_coefs ≤ num_train)){
   num_coefs = user_coefs;
   done = 1;
  }
  else if (user_coefs == 13){
   num_coefs = num_train/3;
   done = 1;
  }
  else
   printf("\nYou need to do at least 1 and at most %d\n",num_train);
}

/********** Create the train_params file for the recogniton phase ********/

fprintf(fparam,"%d\n%d\n%d\n%d\n%d\n",SM_WIDTH,num_coefs,num_train, num_protos, num_class);
fclose(flist);
fclose(fparam);

/************ Create the setup file for the neural network *********/

strcpy(wt_file,"klt.wts");
strcpy(dat_file,"klt.dat");
hid_nodes = 2 * num_coefs;
hid_nodes2 = 0;
fset = fopen("setup.mlp","w");
fprintf
(fset,"%d\n%d\n%d\n%d\n%s                -store weights\n%d\n",NUM_LAYRS,WT_SED,PART_SED,RNDM_SED,wt_file,
MAX_ITS);
fprintf(fset,"%d %d %d %d\n",num_coefs,hid_nodes,hid_nodes2,num_class);
fprintf(fset,"%s                -data\n%d\n%f\n%f\n%f\n%f\n%d\n%f\n%d\n",
dat_file,OUT_INT,ETA_IN,ETA_OUT,ETA_1_2,ALPHA,BAT_SZ,TRAIN_PCT,NORM);
fclose(fset);

/************ Calculate the basis set *****************/

kl_transform("train_list", SM_WIDTH, num_train);

/****** Create the lookup table for the neural network *****/

ftable=fopen("lookup","w");

/************ Create the data file for the neural network ***********/

if ((flist = fopen("train_list", "r")) == NULL)
  printf("\nCollect can't open the training list.\n");

fweights = fopen("klt.dat","w");

fprintf(fweights,"%d\n%d\n",num_coefs,num_class);

for (i=1; i≤num_train; i++){
  fscanf(flist,"%s\n",filename);
  sprintf(command,"%s%s%s","mv ",filename, " training_images");
  net_coefficients(SM_WIDTH, num_coefs, filename, fweights, ftable, num_class);
  system(command);
}
fclose(flist);
fclose(fweights);
system("rm *.rle");
system("rm *.rec");
system("rm *.red");
system("mlp_trn");
printf("\nTRAINING IS COMPLETE\n");
}  /*END TRAIN*/
```

6.4 train_dct.c

```
/*********************************************************
*  train_dct.c
*
*  This program is used to train the system.  The grab routine is first
*  called to collect the training images.  After all images of a
*  particular user have been collected, each of the images are
*  preprocessed (centered and gaussian windowed).
*  The preprocessed images are then used by mdct.c to extract the first
*  64 coefficients from the training images.
*  These coefficients, which are the end result
*  of the training process are stored in train_coefs.  All training images
*  are stored in a folder called training_images for possible use in
*  retraining the system at a later date.
*
*  Written by:  Ken Runyon
*
*  Date:     22 Sep 92
*
*
*********************************************************/

#include <stdio.h>
#include <string.h>
#include "vfc_lib.h"
#include "globals.h"

int   i,
   finished,
   done,
   quit,
   month,
   day,
   select,
   num_protos,
   user_coefs,
   num_coefs=7,
   num_train,
   file_ptr[2];

FILE   *fparam,
   *flist,
   *fweights;

char   command[80],
   image_folder[20],
   u_name[8],
   nu_name[8],
   filename[20],
   waste[2],
   another[4];

main()
{

/********* Make a Training Folder to Hold the Prototypes *******/

system("mkdir training_images");

/**************** Open the Training List File ***************/

if ((flist=fopen("train_list","w+"))==NULL)
   printf("I can't open the train list.\n");

/**************** Open the Training Parameters File ***************/

fparam=fopen("train_params","w");

/********* Prompt User for Number of Prototypes *******/
done = 0;
while (!done){
  printf("\nEnter the number of prototypes to be used for each user <1-20>: ");
  scanf("%d",&num_protos);
  gets(waste);
```

```
  printf("\n");
  if ((num_protos ≤ 20) && (num_protos ≥ 1))
   done = 1;
  else
   printf("\nYou need to do at least 1 and at most 20.\n");
}
/****************** Enter Users Until You're Done ************/

while (!finished){
done = 0;
quit = 0;

/******** Prompt User for User Name ******/

 printf("\nEnter the person's username <8 letters>: ");
 gets(u_name);
 printf("\n");
 while(!done){
   printf("\nThe name you entered was : %s\n",u_name);
   printf("You can either re-enter a name or press return to continue.\n");
   gets(nu_name);
   if (nu_name[0] == '\0')
     done = 1;
   else {
     strcpy(u_name,nu_name);
     nu_name[0] = '\0';
   }
 }

/*********** grab training images of the user **************/

file_ptr[0]=ftell(flist);
 for (i=1; i≤ num_protos; i++){
   num_train = num_train++;
   sprintf(filename,"%s%d",u_name,i);
   seg_grab(filename);
   sprintf(filename,"%s%d%s",u_name,i,".gra");
   fprintf(flist,"%s\n",filename);
 }
 file_ptr[1]=ftell(flist);
 fseek(flist,(file_ptr[0]−file_ptr[1]),1);

/******** Preprocess the training images *****/

for (i=1; i≤num_protos; i++){
   fscanf(flist,"%s\n",filename);
   center(SM_WIDTH,"correlate.ref",filename);
  /* sprintf(filename,"%s%d",u_name,i);
   display(SM_WIDTH,filename,num_protos);
   sprintf(filename,"%s%d%s",u_name,i,".gra");*/

   gwind(SM_WIDTH,filename);
  /* sprintf(filename,"%s%d",u_name,i);
   display(SM_WIDTH,filename,num_protos);
   sprintf(filename,"%s%d%s",u_name,i,".gra");*/

   center(SM_WIDTH,"wind.ref",filename);
   sprintf(filename,"%s%d",u_name,i);
   display(SM_WIDTH,filename,num_protos);
   sprintf(filename,"%s%d%s",u_name,i,".gra");
 }

 printf("\nDo Another User? <y or n>: \n");
 gets(another);
 while (!quit){
 if ((another[0] == 'n') || (another[0] == 'N')){
    finished = 1;
    quit = 1;
 }
 else if ((another[0] == 'y') || (another[0] == 'Y')){
    finished = 0;
    quit = 1;
    }
```

```
else {
    printf("\n\nHit y if you want to enter another user.\nHit n if you're done entering users.  :
");
  gets(another);
  printf("\n");
 }
}
}

fprintf(fparam,"%d\n%d\n%d\n%d\n",SM_WIDTH,num_coefs,num_train, num_protos);
fclose(flist);
fclose(fparam);

/********** Make Coefficients for All Training Images *********/

if ((flist = fopen("train_list", "r")) == NULL)
  printf("\nCollect can't open the training list.\n");

fweights = fopen("train_coefs","w");

for (i=1; i<=num_train; i++){
  fscanf(flist,"%s\n",filename);
  sprintf(command,"%s%s%s","mv ",filename, " training_images");
  dc_transform(SM_WIDTH, filename, fweights ,num_coefs);
  system(command);
}
fclose(flist);
fclose(fweights);
system("rm *.rle");
system("rm *.rec");
system("rm *.red");
printf("\nTRAINING IS COMPLETE\n");
}  /*END TRAIN*/
```

6.5 train.c

```
/*************************************************************
*  train.c
*
*  This program is used to train the KLT\KNN system.
*  The grab routine is called first to collect the training images.  After
*  all images of a particular user have been collected, each of the images
*  are preprocessed (centered and gaussian windowed).
*  The preprocessed images are then used by kl_transform to
*  create an average face and a user determined number of eigen faces.
*  The coefficients module is then called to extract the kl coefficients
*  from the training images.  These coefficients, which are the end result
*  of the training process are stored in train_coefs.  All training images
*  are stored in a folder called training_images for possible use in
*  retraining the system at a later time.
*
*  Written by:  Ken Runyon
*
*  Date:     21 Jul 92 - 31 Aug 92
*
*
*************************************************************/

#include <stdio.h>
#include <string.h>
#include "vfc_lib.h"
#include "globals.h"

int   i,
  finished,
  done,
  quit,
  num_protos,
  user_coefs,
  num_coefs,
  num_train,
```

```
   file_ptr[2];

FILE   *fparam,
   *flist,
   *fweights;

char   command[80],
   u_name[8],
   nu_name[8],
   filename[20],
   waste[2],
   another[4];

 main()
{

/******** Make a Training Folder to Hold the Prototypes *******/

system("mkdir training_images");

/*************** Open the Training List File ***************/

if ((flist=fopen("train_list","w+"))==NULL)
   printf("I can't open the train list.\n");

/*************** Open the Training Parameters File ***************/

fparam=fopen("train_params","w");

/******** Prompt User for Number of Prototypes *******/
done = 0;
while (!done){
  printf("\nEnter the number of prototypes to be used for each user <1-20>: ");
  scanf("%d",&num_protos);
  gets(waste);
  printf("\n");
  if ((num_protos ≤ 20) && (num_protos ≥ 1))
   done = 1;
  else
   printf("\nYou need to do at least 1 and at most 20.\n");
}
/***************** Enter Users Until You're Done *************/

while (!finished){
done = 0;
quit = 0;

/******** Prompt User for User Name *******/

 printf("\nEnter the person's username <8 letters>: ");
 gets(u_name);
 printf("\n");
 while(!done){
   printf("\nThe name you entered was : %s\n",u_name);
   printf("You can either re-enter a name or press return to continue.\n");
   gets(nu_name);
   if (nu_name[0] == '\0')
     done = 1;
   else {
     strcpy(u_name,nu_name);
     nu_name[0] = '\0';
   }
 }

/********** grab training images of the user ***************/

file_ptr[0]=ftell(flist);
 for (i=1; i≤ num_protos; i++){
  num_train = num_train++;
  sprintf(filename,"%s%d",u_name,i);
  seg_grab(filename);
  sprintf(filename,"%s%d%s",u_name,i,".gra");
  fprintf(flist,"%s\n",filename);
 }
```

```
file_ptr[1]=ftell(flist);
fseek(flist,(file_ptr[0]-file_ptr[1]),1);

/******** Preprocess the training images *****/

for (i=1; i<=num_protos; i++){
  fscanf(flist,"%s\n",filename);
  center(SM_WIDTH,"correlate.ref",filename);
  gwind(SM_WIDTH,filename);
  center(SM_WIDTH,"wind.ref",filename);
  sprintf(filename,"%s%d",u_name,i);
  display(SM_WIDTH,filename,num_protos);
  sprintf(filename,"%s%d%s",u_name,i,".gra");
 }

 printf("\nDo Another User? <y or n>: ");
 gets(another);
while (!quit){
 if ((another[0] == 'n') || (another[0] == 'N')){
   finished = 1;
   quit = 1;
 }
 else if ((another[0] == 'y') || (another[0] == 'Y')){
   finished = 0;
   quit = 1;
   }
 else {
     printf("\n\nHit y if you want to enter another user.\nHit n if you're done entering users.  :
");
   gets(another);
   printf("\n");
 }
}
}
/********** Decide how many eigenfaces you need **********/

done = 0;
while (!done){
  printf("\nEnter the number eigenfaces you want to train on <%d>: ",
 num_train/num_protos/3);
  scanf("%d",&user_coefs);
  gets(waste);
  printf("\n");
  if ((user_coefs > 0) && (user_coefs ≤ num_train)){
   num_coefs = user_coefs;
   done = 1;
  }
  else if (user_coefs == 13){
   num_coefs = num_train/num_protos/3;
   done = 1;
   }
  else
   printf("\nYou need to do at least 1 and at most %d\n",num_train);
}

fprintf(fparam,"%d\n%d\n%d\n%d\n",SM_WIDTH,num_coefs,num_train, num_protos);
fclose(flist);
fclose(fparam);

kl_transform("train_list", SM_WIDTH, num_train);

/********** Make Coefficients for All Training Images *********/

if ((flist = fopen("train_list", "r")) == NULL)
  printf("\nCollect can't open the training list.\n");

fweights = fopen("train_coefs","w");

for (i=1; i<=num_train; i++){
  fscanf(flist,"%s\n",filename);
  sprintf(command,"%s%s%s","mv ",filename, " training_images");
  coefficients(SM_WIDTH, num_coefs, filename, fweights);
```

```
  system(command);
}
fclose(flist);
fclose(fweights);
system("rm *.rle");
system("rm *.rec");
system("rm *.red");
printf("\nTRAINING IS COMPLETE\n");
}  /*END TRAIN*/
```

6.6 retrain.c

```
/*************************************************************
*  retrain.c
*
*  This program trains a face recognition system using the pre-existing
*  images stored in the directory of execution.
*  No grabs, no preprocessing, just
*  makes eigenfaces and coefficients.  Training paramaters are selected
*  by the user prompts.  It has to be provided a train_list. This program
*  is used when you want to keep the same training set but change some
*  training parameter such as number of coefficients.
*
*  Written by:  Ken Runyon
*
*  Printed:  10 Sep 92
*
*  Date:     18 Aug 92
*
*
*************************************************************/
#include <stdio.h>
#include <string.h>

int   i,
   done,
   width=32,
   user_coefs,
   num_coefs,
   num_train;

FILE   *fparam,
   *flist,
   *fweights;

char   command[80],
   filename[20],
   waste[2];

extern kl_transform();
extern void coefficients();

 main()
{

/*************** Open the Training List File ***************/

if ((flist=fopen("train_list","a"))== NULL) {
   printf("I can't open the train list.\n");
   exit(-1);
}

/*************** Open the Training Parameters File ***************/

if((fparam = fopen("train_params","r"))== NULL){
  printf("I can't open the parameter file.");
  exit(-1);
}
/*************** Open the trainlist ***************/

if((flist = fopen("train_list","r"))== NULL){
  printf("I can't open the training list.");
```

```
  exit(-1);
}

/********* Count the number of training images ******/

while (!feof(flist)){
   fscanf(flist,"%s\n",filename);
   num_train++;
}
printf("\nRetraining on %d faces:\n",num_train);

fclose(flist);

/*********** Decide how many eigenfaces you need **********/

while (!done){
  printf("\nEnter the number eigenfaces you want to train on <%d>: ",num_train/3);
  scanf("%d",&user_coefs);
  gets(waste);
  printf("\n");
  if ((user_coefs > 0) && (user_coefs ≤ num_train)){
   num_coefs = user coefs;
   done = 1;
  }
  else if (user_coefs == 13){
   num_coefs = num_train/3;
   done = 1;
  }
  else
   printf("You need to do at least 1 and at most %d\n",num_train);
}

fprintf(fparam,"%d\n%d\n%d\n",width,num_coefs,num_train);
fclose(fparam);
system("cp training_images/* .");

/*kl_transform("train_list", width, num_train); */

/************ Make Coefficients for All Training Images *********/

if ((flist = fopen("train_list","r")) == NULL) {
  printf("I can't open the training list.\n");
  exit(-1);
}

system("rm train_coefs");
fweights = fopen("train_coefs","w");

for (i=1; i≤num_train; i++){
  fscanf(flist,"%s\n",filename);
  sprintf(command,"%s%s%s","mv ",filename," training_images");
  coefficients(width, num_coefs, filename, fweights);
  system(command);
}
fclose(flist);
}  /*END TRAIN*/
```

6.7 add_usr.c

```
/*************************)*****************************
*  add_usr.c
*
*  This program is used to add a user to the set of training faces.
*  Images of the new user are taken and preprocessed.  The new images
*  are added to the train_list. A new average face and a new set of eigen-
*  faces are then created using the new images and the pre-existing images
*  stored in the training_images folder.
*  Coefficients are then extracted for all of the images and a new
*  train_coefs file is written.
*
*  Written by:  Ken Runyon
```

```
*
*  Date:      21 Jul 92 - 25 Aug 92
*
*
************************************************************/
#include <stdio.h>
#include <string.h>

int    i,
   finished,
   done = 1,
   quit = 0,
   month,
   day,
   width,
   num_protos,
   user_coefs,
   num_coefs,
   num_train,
   file_ptr[2];

FILE   *fparam,
   *flist,
   *fweights;

char   command[80],
   u_name[8],
   nu_name[8],
   filename[20],
   waste[2],
   another[4];

extern void seg_grab();
extern void display();
extern void center();
extern void gwind();
extern void coefficients();

 main()
{

/*************** Open the Training List File ***************/

if ((flist=fopen("train_list","a"))==NULL) {
   printf("I can't open the train list.\n");
   exit(-1);
}

/*************** Open the Training Parameters File ***************/

if((fparam = fopen("train_params","r"))== NULL){
  printf("I can't open the parameter file.");
  exit(-1);
}
fscanf(fparam,"%d",&width);
fscanf(fparam,"%d",&num_coefs);
fscanf(fparam,"%d",&num_train);
fscanf(fparam,"%d",&num_protos);
fclose(fparam);

/***************** Enter Users Until You're Done *************/

while (!finished){
done = 0;
quit = 0;

/********* Prompt User for User Name *******/

 printf("Enter the person's username <8 letters>: ");
 gets(u_name);
 printf("\n");
 while(done == 0){
   printf("The name you entered was : %s\n",u_name);
   printf("You can either re-enter a name or press return to continue.\n");
   gets(nu_name);
```

```
  if (nu_name[0] == '\0')
    done = 1;
  else {
    strcpy(u_name,nu_name);
    nu_name[0] = '\0';
  }
}

/********** grab training images of the user **********/

file_ptr[0]=ftell(flist);
for (i=1; i<= num_protos; i++){
  num_train = num_train++;
  sprintf(filename,"%s%d",u_name,i);
  seg_grab(filename);
  sprintf(filename,"%s%d%s",u_name,i,".gra");
  fprintf(flist,"%s\n",filename);
}
file_ptr[1]=ftell(flist);
fseek(flist,(file_ptr[0]-file_ptr[1]),1);

/********* Preprocess the training images ******/

for (i=1; i<=num_protos; i++){
  fscanf(flist,"%s\n",filename);
  center(width,"correlate.ref",filename);

  gwind(width,filename);

  center(width,"wind.ref",filename);
  sprintf(filename,"%s%d",u_name,i);
  display(width,filename,num_protos);
  sprintf(filename,"%s%d%s",u_name,i,".gra");
}
fseek(flist,(file_ptr[1]-file_ptr[0]),1);
printf("\nDo Another User? <y or n>: \n");
gets(another);
while (quit == 0){
 if ((another[0] == 'n') || (another[0] == 'N')){
   finished = 1;
   quit = 1;
 }
 else if ((another[0] == 'y') || (another[0] == 'Y')){
   finished = 0;
   quit = 1;
   }
 else {
     printf("\n\nHit y if you want to enter another user.\nHit n if you're done entering users.  :
");
   gets(another);
   printf("\n");
 }
}
}
fclose(flist);
/********** Decide how many eigenfaces you need **********/

while (done == 1){
  printf("Enter the number eigenfaces you want to train on <%d>: ",num_train/3);
  scanf("%d",&user_coefs);
  gets(waste);
  printf("\n");
  if ((user_coefs > 0) && (user_coefs <= num_train)){
   num_coefs = user_coefs;
   done = 0;
  }
  else if (user_coefs == 13){
   num_coefs = num_train/3;
   done = 0;
  }
  else
   printf("You need to do at least 1 and at most %d\n",num_train);
```

```
}
if((fparam = fopen("train_params","w"))== NULL){
  printf("I can't open the parameter file.");
  exit(-1);
}

fprintf(fparam,"%d\n%d\n%d\n%d\n",width,num_coefs,num_train,num_protos);
fclose(fparam);
system("cp training_images/*.gra .");
kl_transform("train_list", width, num_train);

/************ Make Coefficients for All Training Images *********/

if ((flist = fopen("train_list", "r")) == NULL) {
  printf("I can't open the training list.\n");
  exit(-1);
}

fweights = fopen("train_coefs","w");

for (i=1; i<=num_train; i++){
  fscanf(flist,"%s\n",filename);
  sprintf(command,"%s%s%s","mv ",filename, " training_images");
  coefficients(width, num_coefs, filename, fweights);
  system(command);
}
fclose(flist);
fclose(fweights);
system("rm *.rle");
system("rm *.red");
system("rm *.rec");
}  /*END TRAIN*/
```

6.8 verify.c

```
/*********************************************************
*   Name:       verify.c
*
*
*   Description:  This program performs face recognition.  The program
*         grabs an image of the person sitting in front of the
*         camera, processes that image, extracts the KLT
*         coefficients and finds the closest
*         match from the faces in the training set using the
*         K-nearest neighbor.
*
*   Written by:  Kenneth Runyon
*
*   Date:       8 July 92 - 31 Aug 92
*
*
*********************************************************/
#include <stdio.h>

int    dimension,
   num_coefs,
   num_train_faces,
   done,
   num_protos;

FILE   *fparam,
   *fweights;

char   answer[2],waste[2];

main()
{

/****** Grab a test image to align the user ******/
```

```
while(!done) {
   grab(64);
   system("bin2gray user.red test.gra");
   system("rm user.red");
   display(64, "test",1);
   printf("Is your face completely in the picture? <y or n> :");
    while(((answer[0]≠'y') && (answer[0]≠'Y')) && ((answer[0]≠'n') &&
 (answer[0]≠'N'))){
      gets(answer);
      printf("\n");
      if (answer[0] == 'y' || answer[0] == 'Y'){
         done = 1;
           }
      else if(answer[0] == 'n' || answer[0] == 'N'){
         printf("answer = %c\n",answer[0]);
         done = 0;
          }
      else  {
            printf
("Enter 'y' if your whole face is in the picture.\n");
            printf
("Enter 'n' if your whole face is not in the picture.\n\n");
       }
   }
answer[0] = 0;
}
system("rm test.gra");

/*****************************************************
*   read the parameters from train_params file
*****************************************************/
if((fparam = fopen("train_params","r"))== NULL){
  printf("I can't open the parameter file.");
  exit(-1);
}
fscanf(fparam,"%d",&dimension);
fscanf(fparam,"%d",&num_coefs);
fscanf(fparam,"%d",&num_train_faces);
fscanf(fparam,"%d",&num_protos);
fclose(fparam);

/****** grab a test image ******/
seg_grab("user");

/****** preprocess the test image ******/

center(dimension,"correlate.ref","user.gra");
gwind(dimension,"user.gra");
center(dimension,"wind.ref","user.gra");
display(dimension, "user",1);

/****** pull the kl coefficients and store them ******/

fweights = fopen("test_coefs","w");
coefficients(dimension, num_coefs,"user.gra",fweights);
fclose(fweights);

/****** find the best matching training face ******/

k_nearest(num_protos,num_coefs,num_train_faces);
system("rm test_coefs");

/****** remove trash files *******/

system("rm *.rle");
system("rm *.red");
system("rm *.rec");

} /*** end verify.c ***/
```

6.9 verify_net.c

```
/*********************************************************
*   Name:       verify_net.c
*
*
*   Description:   This program performs face recognition.  The program
*          grabs an image of the person sitting in front of the
*          camera, processes that image, extracts the KLT
*          coefficients and finds the closest
*          match from the faces in the training set.
*
*   Written by:   Kenneth Runyon
*
*   Date:       8 July 92 - 31 Aug 92
*
*
*********************************************************/
#include <stdio.h>
#define   NUM_LAYRS   2
#define   WT_SED      1918940490
#define PART_SED   1191645590
#define   RNDM_SED   123456789
#define MAX_ITS      3000
#define   OUT_INT      100
#define   ETA_IN      0.15
#define   ETA_OUT      0.3
#define ETA_1_2      0.0
#define   ALPHA      0.5
#define   BAT_SZ      1
#define   TRAIN_PCT   1.0
#define   NORM      1

int    dimension,
   num_coefs,
   num_train_faces,
   done,
   num_class,
   num_protos;

FILE    *fparam,
   *fweights,
   *fset,
   *ftable;

char   wt_file[10],
   dat_file[10],
   hid_nodes,
   hid_nodes2,
   answer[2],
   waste[2];

extern void grab();
extern void ver_grab();
extern void display();
extern void center();
extern void gwind();
extern void net_coefficients();
extern void k_nearest();

main()
{
/****** Grab a test image to align the user ******/

while(!done) {
   grab(64);
   system("bin2gray user.red test.gra");
   system("rm user.red");
   display(64, "test",1);
   printf("Is your face completely in the picture? <y or n> :");
    while(((answer[0]≠'y') && (answer[0]≠'Y')) && ((answer[0]≠'n') &&
(answer[0]≠'N'))){
      gets(answer);
      printf("\n");
      if (answer[0] == 'y' || answer[0] == 'Y'){
```

```
        done = 1;
          }
      else if(answer[0] == 'n' || answer[0] == 'N'){
        printf("answer = %c\n",answer[0]);
        done = 0;
          }
      else  {
            printf
("Enter 'y' if your whole face is in the picture.\n");
            printf
("Enter 'n' if your whole face is not in the picture.\n\n");
        }
    }
answer[0] = 0;
}
system("rm test.gra");

/*****************************************************
*   read the parameters from train_params file
*****************************************************/
if((fparam = fopen("train_params","r"))== NULL){
  printf("I can't open the parameter file.");
  exit(-1);
}
fscanf(fparam,"%d",&dimension);
fscanf(fparam,"%d",&num_coefs);
fscanf(fparam,"%d",&num_train_faces);
fscanf(fparam,"%d",&num_protos);
fscanf(fparam,"%d",&num_class);
fclose(fparam);
/********** Create the setup file for the neural network *********/

strcpy(wt_file,"klt.wts");
strcpy(dat_file,"klt.dat");
hid_nodes = 2 * num_coefs;
hid_nodes2 = 0;
fset = fopen("setup.mlp","w");
fprintf
(fset,"%d\n%d\n%d\n%d\n%s                -store weights\n%d\n",NUM_LAYRS,WT_SED,PART_SED,RNDM_SED,wt_file,
MAX_ITS);
fprintf(fset,"%d %d %d %d\n",num_coefs,hid_nodes,hid_nodes2,num_class);
fprintf(fset,"%s              -data\n%d\n%f\n%f\n%f\n%d\n%f\n%d\n",
dat_file,OUT_INT,ETA_IN,ETA_OUT,ETA_1_2,ALPHA,BAT_SZ,TRAIN_PCT,NORM);
fclose(fset);

/****** grab a test image *****/
ver_grab();

/****** preprocess the test image *****/

center(dimension,"correlate.ref","user.gra");
gwind(dimension,"user.gra");
center(dimension,"wind.ref","user.gra");
display(dimension, "user",1);

/****** create the data file and store the kl coefficients *****/

ftable=fopen("waste","w");
fweights = fopen("klt.dat","w");
fprintf(fweights,"%d\n%d\n",num_coefs,num_class);
net_coefficients(dimension, num_coefs,"user.gra", fweights, ftable, num_class);
fclose(fweights);

/****** find the best matching training face *****/

system("mlp_tst");


/****** remove trash files ******/

/*system("rm test_coefs");*/
system("rm *.rle");
system("rm *.red");
system("rm *.rec");
system("rm waste");

} /*** end verify.c ***/
```

6.10 verify_dct.c

```
/*********************************************************
*   Name:       verify_dct.c
*
*
*   Description:   This program performs face recognition.  The program
*         grabs an image of the person sitting in front of the
*         camera, processes that image, extracts the DCT
*         coefficients and finds the closest
*         match from the faces in the training set.
*
*   Written by:   Kenneth Runyon
*
*
*   Date:      Sep 92
*
*   To be done:   If(pers is user) then login else logout
*
*********************************************************/
#include <stdio.h>
#define  SQ(A) (A)*(A)

int    dimension,
   coef_rows,
   num_coefs,
   num_train_faces,
   done,
   num_protos;

FILE    *fparam,
   *fweights;

char   answer[2],waste[2];

extern void new_grab();
extern void display();
extern void center();
extern void gwind();
extern void dc_transform();
extern void vote();

main()
{

/****** Grab an image to align the user ******/

while(!done) {
   grab(64);
   system("bin2gray user.red test.gra");
   system("rm user.red");
   display(64, "test",1);
   printf("Is your face completely in the picture? <y or n> :");
    while(((answer[0]≠'y') && (answer[0]≠'Y')) && ((answer[0]≠'n') &&
(answer[0]≠'N'))){
      gets(answer);
      printf("\n");
      if (answer[0] == 'y' || answer[0] == 'Y'){
         done = 1;
           }
      else if(answer[0] == 'n' || answer[0] == 'N'){
         printf("answer = %c\n",answer[0]);
         done = 0;
          }
      else  {
            printf("Enter 'y' if your whole face is in the picture.\n");
            printf("Enter 'n' if your whole face is not in the picture.\n\n");
       }        /*end of else*/
   }          /*end of while (answer) */
answer[0] = 0;
}             /*end of while(done)*/
system("rm test.gra");
```

```
/*******************************************************
*  read the parameters from train_params file
*******************************************************/
if((fparam = fopen("train_params","r"))== NULL){
  printf("I can't open the parameter file.");
  exit(-1);
}
fscanf(fparam,"%d",&dimension);
fscanf(fparam,"%d",&coef_rows);
num_coefs = SQ(coef_rows+1);
fscanf(fparam,"%d",&num_train_faces);
fscanf(fparam,"%d",&num_protos);
fclose(fparam);

/****** Grab the test image ******/

ver_grab();

/****** Preprocess the test image ******/

center(dimension,"correlate.ref","user.gra");
gwind(dimension,"user.gra");
center(dimension,"wind.ref","user.gra");
display(dimension, "user",1);

/****** Write out the DCT coefficients ******/

fweights = fopen("test_coefs","w");
dc_transform(dimension,"user.gra",fweights,coef_rows);
fclose(fweights);

/****** Find the closest matching training face ******/

k_nearest(num_protos,num_coefs,num_train_faces);

/****** Remove the trash files ******/

system("rm test_coefs");
system("rm *.rle");
system("rm *.red");
system("rm *.rec");
}/*** end verify_dct.c ***/
```

6.11 klt.c

```
/*********************************************************************
  NAME: klt.c

  DATE: 24 June 1991

  DESCRIPTION: This program calculates the basis set of eigenfaces for
    a given training set as well as the average face.

  Modified by:  Ken Runyon

  Printed:  10 Sep 92

  Date:  30 Jul 92

  Changes:  Modularized the program to a procedure.  Modified the
    interface to pass training parameters.

*********************************************************************/

#include <stdio.h>
#include <math.h>
#include <string.h>

#define SQ(A)  (A*A)

char  train_list[];
```

```
int   dimension,
   num_train;

void kl_transform(train_list, dimension, num_train)
{

FILE    *train, *facein,  *fout;
FILE    *face_avg, *tempfile, *fevex, *feval;
int    i,j, N, k, M, nrot, atoi();
float     **matrix(),  *vector(), **A, **A_trans, **u, **L, **v,  *average_face;
float    *d, temp, *mag;
void    free_vector(), free_matrix(), eigsrt(), jacobi(), mat_col_mag();
char   filename[81], h,l;


/****************** Set Up Files ************************/

printf("Creating Eigenfaces.\n\n");
N = dimension * dimension;
if ((train = fopen(train_list, "r")) == NULL)
{
printf("I can't open the training list");
exit(-1);
}

M = num_train;

/****** dynamically allocate memory ******/

A_trans = matrix(1,M,1,N);
A = matrix(1,N,1,M);
average_face = vector(1,N);
L = matrix(1,M,1,M);
d = vector(1,M);
v = matrix(1,M,1,M);
mag = vector(1, M);


/****** initalize matrix and vectors ******/

for(j=1;j≤M;j++)
   for(i=1;i≤N;i++) {
      A_trans[j][i]=A[i][j]=average_face[i]=0.0;
   }

printf("The files being trained on are :\n\n");

for(k=1; k≤M; k++){

   fscanf(train, "%s\n", filename);
      printf("\t\t%s\n", filename);

   facein = fopen(filename, "r");


    for(j=1;j≤N;j++){
      fscanf(facein,"%f\n",&A[j][k]);
        }
    fclose(facein);
}

/****** Normalizing Data by dividing by 255 ******/

 for(j=1;j≤M;j++)
   for(i=1;i≤N;i++) {
      A[i][j]=A[i][j]/255;
   }

/**********Calculate Average Face ********************/

/*printf("!!! CALCULATE AVERAGE FACE\n ");*/
face_avg=fopen("avg_face.dat","w");
for(i=1;i≤N;i++){
   temp=0.0;
```

```
   for(j=1;j<=M;j++)
      temp=temp+A[i][j];
   average_face[i]=temp/M;
   fprintf(face_avg,"%f\n", average_face[i]);
}
fclose(face_avg);

/****** printf("!!! SUBTRACTING OFF AVERAGE FACE \n"); ******/

for(j=1;j<=M;j++)
   for(i=1;i<=N;i++){
      A[i][j]=A[i][j]-average_face[i];
         }

/****** CREATING A TRANSPOSE ******/

/* printf("!!!  CREATING TRANSPOSE MATIX  \n");*/

for(j=1;j<=M;j++)
   for(i=1;i<=N;i++){
      A_trans[j][i]=A[i][j];
   }


/****** fout=fopen("1.dat","w");******/

/* printf("!!!  Multiplying A trans and A to get L:\n"); */

for(i=1;i<=M;i++)
   for(j=1;j<=M;j++) {
      temp=0.0;
      for(k=1;k<=N;k++){
         temp=temp+A_trans[i][k]*A[k][j];
      }
      /******fprintf(fout,"%f\n", temp);******/

      L[i][j]=temp;
      /****** printf("!!!  Writing Output \n"); ******/
   }

/****** ("!!!  FREE MATRIX A_TRANS \n");  ******/

free_matrix(A_trans, 1, M, 1, N);

/*printf("!!! doing jacobian of L \n");  */

jacobi(L,M,d,v, &nrot);

eigsrt(d,v,M);

/**********
*feval=fopen("eigen_val","w");
*for(j=1;j<=M;j++) {
*      fprintf(feval,"%f\n", d[j]);
*   }
*fclose(feval);
**********/


/********* printf("!!! Writing eigenvectors \n");
*
8fevex=fopen("eigen_vec","w");
8for(i=1;i<=M;i++){
*   for(j=1;j<=M;j++) {
*      fprintf(fevex,"%f\n", v[j][i]);
*            }
*   }
*   fclose(fevex);
*********/
```

```
/****** printf("!!! Initializing Eigenface Matrix \n"); ******/

u = matrix(1,N,1,M);
for(k=1;k≤M; k++)
   for(j=1;j≤N; j++)
      u[j][k]=0.0;


/*printf("!!! Calcualting eigenface \n"); */

for(k=1;k≤M; k++){
   for(i=1;i≤M; i++){
      for(j=1;j≤N; j++)
         u[j][k]=v[i][k]*A[j][i]+u[j][k];
   }
}


/****** finding magnitude of eigenface ******/

/*printf("!!! Opening train.out file for Eigenfaces \n"); */
tempfile = fopen("train.out","w");

h=48;
l=48;
strcpy(filename, "eigenface");

for(k=1; k≤M; k++){

   if(l ≠ 57) l++;
         else{
            l=48;
            h++;
         }


   fprintf(tempfile,"%s", filename);

   fprintf(tempfile,"%c%c",h,l);

   fprintf(tempfile,"%s\n",".dat");

}
fclose(tempfile);


/*printf("!!! Writing eigenface \n");*/

tempfile = fopen("train.out","r");

for(k=1; k≤M; k++){

   fscanf(tempfile, "%s\n", filename);

   facein = fopen(filename, "w");

   /****** printf("%s\n", filename); ******/
   /****** fprintf(facein, "%f\n", mag[k]); ******/

    for(j=1;j≤N;j++){
      fprintf(facein,"%g\n", u[j][k]);

         }
    fclose(facein);
}


fclose(tempfile);

/******printf("!!!  FREEING A MATRIX \n");*****/

free_matrix(A,1,N,1,M);
```

```
free_matrix(u,1,N,1,M);

}
void mat_col_mag(u, N, M, mag)
float **u, mag[];
int N,M;
{float  b;
int k, j;
double   sqrt ();

for(k=1; k≤M; k++) {
   b=0;
   for(j=1; j≤N; j++)
      b=u[j][k] * u[j][k] + b;
   mag[k] = sqrt( (double) b);
}
}
```

6.12 seg_grab.c

```
/*
 * File:   seg_grab.c
 * Created:   July 1992
 * By:      Kevin Gay
 *
 * Purpose:   Uses the VideoPix Tool to grab segmented images.
 *
 * Assumes:
 *
 * Modified:   12 Aug 92
 *
 * By:      Ken Runyon
 *
 * Why:      I changed the linked list of images to return only one good
      image instead of returning every attempt.  This grab is used for
      training and recognition routines.
*/

#include <stdio.h>
#include <sys/types.h>
#include "vfc_lib.h"
#include "globals.h"

extern struct head_ptrs      *z_segment();
extern int          z_store_image();

void seg_grab(filename)

char   filename[];

{
   u_char          *face, *ptr;
   register int       i;
   struct head_ptrs   *temp_ptr, *face_ptrs;
   char          okay, tryagain = 'y',
            storefile[30],command[80],
            waste[2];
   int          done=0;

/*** will store image as a "red" file ***/

sprintf(storefile,"%s%s",filename,".red");

/*** loop to detect motion and segment face ***/

while((tryagain == 'Y')||(tryagain == 'y')){
   done = 0;
   printf("Please look at camera until you hear a beep.\n");

/*** assign segmented images to face_ptrs linked list ***/

    face_ptrs = (struct head_ptrs *)z_segment();
```

```
    if(face_ptrs == NULL)
      printf("Trouble getting images\n");
    else {
      while(face_ptrs→next ≠ 0) {
        face_ptrs = face_ptrs→next;
      }
        if(z_store_image(face_ptrs→head,storefile,SM_SIZE)<0)
          fprintf(stderr,"Unable to write to file\n");
        sprintf(command,"%s%s%s%s%s","bin2gray ",filename,
          ".red ",filename,".gra");
        system(command);
        display(32,filename,1);
        printf("\nIs this picture okay? <y or n> ");
        while(!done){
          scanf("%c",&okay);
          gets(waste);
          if (okay == 'n' || okay == 'N'){
            free(face_ptrs);
            tryagain ='y';
            done = 1;
          }
          else if(okay == 'y' || okay == 'Y'){
            free(face_ptrs);
            sprintf(command,"%s%s%s","rm ",filename,".red");
            system(command);
            tryagain = 'n';
            done = 1;
          }
          else printf("\nIs all of your head in the picture? <y or n>");
          }
    }
   }
} /* end of seg_grav*/
```

6.13 center.c

```
/*********************************************************
*   Name:       center.c
*
*   Description:   This program correlates the input image with a
*          reference image and shifts the input image so
*          that it overlays the reference image as much as
*          possible.  This is the current way to make the KL
*          transform invariant to shift.
*
*   Written by:   Pedro Suarez (at least that's who I got it from)
*
*   Date:       Summer 91
*
*   Modified by:   Ken Runyon
*
*   Date:       15 Jul 92
*
*   Modification:   Converted it forom stand alone c to a module,
*          changed the output file to write back to the input,
*       pulled our rescale(); as a seperate procedure. *********************************************************/

#include <stdio.h>
#include <math.h>
#define SQR(a) (a)*(a)
#define loopi(A) for(i=0;i<(A);i++)
#define loopj(A) for(j=0;j<(A);j++)

void center(Row, refile, imagefile)
   int   Row;

   char   refile[],
      imagefile[];
{
int   x,y,i,j,
```

```
  Col = Row,
  location[3];

float  *x1,
  *x2,
  *vector(),
  **matrix(),
  **image1,
  **image2;

FILE  *fout,
  *dat_file1,
  *dat_file2,
  *out_file;

void    fourn();
void    Correlate();
void     max_find();
void     C_late3();
void     shift();
void   rescale();

printf("\nCentering %s\n",imagefile);

/****************** Set Up Files ***********************/

if ((dat_file1 = fopen(refile, "r")) == NULL)
{
printf("I can't open the reference image");
exit(-1);
}

if ((dat_file2 = fopen(imagefile, "r")) == NULL)
{
printf("I can't open the input image");
exit(-1);
}

/****** allocate matrices for images ******/

image1 = matrix(0, Row-1, 0, Col-1);
image2 = matrix(0, Row-1, 0, Col-1);

/****** read the reference face into the 1st matrix ******/

for(y=0; y<Row; y++)
  for(x=0; x<Col; x++)
    fscanf(dat_file1, "%f\n",&image1[x][y]); /** (takes care of initialization) **/
fclose(dat_file1);

/****** read the input face into the 2nd matrix ******/

for(y=0; y<Row; y++)
 for(x=0; x<Col; x++)
   fscanf(dat_file2, "%f\n",&image2[x][y]);
fclose(dat_file2);

/****** Open image file as output file ******/

if ((out_file = fopen(imagefile, "w")) == NULL)
{
printf("I can't open the output file");
exit(-1);
}

/****** allocate two vectors and initialize them ******/

x1 = vector(0,2*Row*Col-1);
x2 = vector(0,2*Row*Col-1);

loopi(Row*Col){
  x1[2*i] = x1[2*i+1] = 0.0;
  x2[2*i] = x2[2*i+1] = 0.0;
   }
```

```
/****** Put the images into every other space of the column vectors ******/

loopi(Row) {
  loopj(Col) (float)  x1[2*(i*Col+j)] = image1[j][i];
  loopj(Col)  (float) x2[2*(i*Col+j)] = image2[j][i];
}

/****** Finds the peak correlation value ******/

C_late3(x1, x2, Row, Col, location);
if(location[0]>(Col/8)) location[0] = -(Col - location[0]);
if(location[1]>(Row/8)) location[1] = -(Row - location[1]);

shift(image2, Row, Col, location);

rescale(image2, Row, Col);

for (y = 0; y < Row; y++)
  for (x = 0; x < Col; x++)
    fprintf(out_file, "%4.0f\n ",image2[x][y]);
fclose(out_file);
free_matrix(image1,0, Row-1, 0, Col-1);
free_matrix(image2,0, Row-1, 0, Col-1);
free_vector(x1,0,2*Row*Col-1);
free_vector(x2,0,2*Row*Col-1);
}
/**********************THIS IS THE END OF CENTER.C*********************/

/*****************************************************************************
   NAME: C_late3
   DESCRIPTION: This routine determines the number of rows and columns that
        the input image needs to be shifted by to be centered on the
      reference
*****************************************************************************/

void C_late3(x1, x2, Row, Col, location)
int Row, Col;
int location[];
float x1[], x2[];
{

 FILE   *out_file;
 int    n[2], i, j;
 float   *output,
    temp,
   **matrix(),
   **mat_output,
   *vector();

/****** Allocate Memory for Arrays ******/

output = vector(0,2*Row*Col-1);
mat_output = matrix(0, Row-1, 0, Col-1);

/****** Assign Initial Array Values ******/

n[0] = Col;
n[1] = Row;

Correlate(x1, x2, output, n, Row, Col);

/****** Store The Magnitude Results ******/

loopi(Row)
  loopj(Col) {
    temp = sqrt((double)SQR(output[2*(i*Col+j)])
        +(double)SQR(output[2*(i*Col+j)+1]));
    mat_output[j][i] =  (float) temp;
  }
max_find(mat_output, Row, Col, location);
free_vector(output,0,2*Row*Col-1);
free_matrix(mat_output,0, Row-1, 0, Col-1);
}
/****** This is the end of C_late3 ******/
```

```
/*****************************************************************************
   NAME: Correlate
   DESCRIPTION: This routine shifts the input image over the reference and
        calculates a corelation value for each location
*****************************************************************************/

void Correlate(input1, input2, output, n, Row, Col)
float   input1[],
   input2[],
   output[];
int   n[],
   Row,
   Col;
{
int   i;
float   *temp1,
   *temp2;

temp1 = vector(0,2*Row*Col-1);
temp2 = vector(0,2*Row*Col-1);

loopi(2*Row*Col){
  temp1[i] = input1[i];
  temp2[i] = input2[i];
  }

/****** Take Fourier Transform of Input Functions ******/

fourn(temp1-1, n-1, 2, 1);
fourn(temp2-1, n-1, 2, 1);

/****** Conjugate One of The Fourier Transforms ******/

loopi(Row*Col)
   temp2[2*i+1] = -temp2[2*i+1];

/****** Multiply Fourier Transforms Together ******/

loopi(Row*Col){
  output[2*i] = temp1[2*i]*temp2[2*i] - temp1[2*i+1]*temp2[2*i+1];  /* Real */
  output[2*i+1] = temp1[2*i]*temp2[2*i+1] + temp2[2*i]*temp1[2*i+1]; /* Imaginary */
}

/****** Take Inverse Transform to obtain Correlation ******/

fourn(output-1, n-1, 2, -1);

/****** Rescale to get proper magnitude ******/

loopi(2*Row*Col)
  output[i] /= Row*Col;

/*****************************************************************************
The result of the correlation is that thefirst element of the output
matrix is for zero shift, the next element for shift one to the right and
so on.  This puts results into a format which  humans can understand.
*****************************************************************************/

/** Free up the memory when finished **/

free_vector(temp1,0,2*Row*Col-1);
free_vector(temp2,0,2*Row*Col-1);
}
/********************* End of Correlate Routine *********************/

/*****************************************************************************
   NAME: max_find
   DESCRIPTION: This routine finds the max value in a vector
*****************************************************************************/

void max_find(mat_output, row, col, location)

float   **mat_output;
int   row,
   col,
```

```
  location[];
{
int    i, j,
  count,
  temp_i,
  temp_j;
float  max;

/****** Check for the max and min value in the data ******/

max = mat_output[0][0];
count=0;

for(j=0; j<row; j++)
  for(i=0; i<col; i++){
    if(mat_output[i][j]>max) {
   max = mat_output[i][j];
   temp_i = i; temp_j = j;
    }
    count++;
  }
/*printf("\n1st_shift=%d\n",temp_i);
printf("\n2nd_shift=%d\n",temp_j);*/
 location[0]=temp_i;  location[1]=temp_j;
  if((location[0]>col/8)||(location[0]<(0 - col/8))) location[0]=0;
  if((location[1]>row/8)||(location[1]<(0 - row/8))) location[1]=0;
}

/*****************************************************************************
  NAME: shift
  DESCRIPTION: This routine takes a image shifts
*****************************************************************************/

void shift(image, Row, Col, location)

float  **image;
int Row, Col;
int location[];
{
int    **temp_image,
  i,x,y,
  n,k,
  new_row,
  new_col,
  xshift,
  yshift,
  abs();
float   **matrix(),
  **shifted_image;

xshift = location[0];
yshift = location[1];
/*printf("\nxshift = %d\n",xshift);
printf("\nyshift = %d\n",yshift);*/

new_row= (int) Row+ 2 * abs(yshift);
new_col=(int)  Col+ 2 * abs(xshift);

shifted_image = matrix( -abs(xshift), new_col, -abs(yshift), new_row);

/*** initialize matrix ***/

for(y= - abs(yshift); y<new_row; y++)
  for(x= - abs(xshift); x< new_col; x++)
       shifted_image[x][y]=127.0;

for(y=0; y<Row; y++)
    for(x=0; x<Col; x++){
     shifted_image[x+xshift][y+yshift] = image[x][y];
  }


for (y = 0; y < Row; y++)
```

```
   for (x = 0; x < Col; x++)
      image[x][y]=shifted_image[x][y];

free_matrix(shifted_image, Row, Col, location);

}
```

6.14 coefficients.c

```
/*********************************************************
*   Name:       coefficients.c
*
*
*   Description:  This program maps a test face onto the set of
*         eigenfaces and stores the KL coefficients in
*         test_set.ng.
*
*   Written by:  Pedro Suarez   (Originally recon.c)
*
*   Date:       24 July 91
*
*   Modified by:  Ken Runyon   (Chopped off reconstruction)
*
*   Date:       22 Jun 92
*
*   Modifications:  I decided we didn't need to actually reconstruct and
*         store a face.  I also made the stand alone program
*         into a module which is called by thesis.
*********************************************************/
#include <stdio.h>
#include <math.h>

void coefficients(dimension, num_coefs, infilename, outfile)
   int   dimension,
      num_coefs;

   char   infilename[];

   FILE   *outfile;
{
FILE    *face1, *eigenin, *fweights, *train, *face_avg;
int     i,j, N, M, atoi();
float     *vector(), **matrix(), **free_matrix(), *average_face, **u, *pedro, *reconface;
float     *w, *I;
char    filename[81], *strcpy(), user[8], ext[10];

printf("\nPulling Coefficients for %s\n",infilename);

/****************** Set Up Files ***********************/

/*** Open Test Face ***/

if ((face1=fopen(infilename,"r")) == NULL){
 printf("I can't open the input file");
 exit(-1);
 }

 /*** Open Avg Face ***/

if ((face_avg=fopen("avg_face.dat","r"))== NULL){
 printf("I can't open avg_face.dat.");
 exit(-1);
 }

/****** set up matrices ******/

N = dimension * dimension;
M = num_cocfs;

u = matrix(1,N, 1,M);
pedro = vector(1, N);
```

```
average_face = vector(1, N);
reconface = vector(1, N);   /*** DO I NEED THIS? ***/
w = vector(1, M);
I = vector(1, N);


/****** Initalize Matrices ******/

for(j=1; j≤M; j++)
   for(i=1;i<N;i++)
   w[j]=u[i][j]=I[i]=pedro[i]=reconface[i]=average_face[i]=0.0;

/********* Load the Test Face into the Pedro Vector ********/

 for(j=1;j≤N;j++)
   fscanf(face1,"%f\n", &pedro[j]);
 fclose(face1);

 /********* Load the Average Face into the Average_Face Vector ***/

for(j=1;j≤N;j++)
    fscanf(face_avg, "%f\n", &average_face[j]);
fclose(face_avg);

/****** Load the Eigenfaces into Matrix U ******/

train = fopen( "train.out", "r");

for(j=1; j≤M; j++){
   fscanf(train, "%s\n", filename);
   eigenin = fopen(filename, "r");
   for(i=1;i≤N;i++){
      fscanf(eigenin,"%f\n",&u[i][j]);
   }
   fclose(eigenin);
}
fclose(train);

/****** Subtract the Average Face from the Test Face ******/

   for(i=1;i≤N;i++)
      I[i]= pedro[i] − average_face[i];

/******  Calculate the KL Coefficients ******/

for(j=1; j≤M; j++)
   for(i=1; i<N; i++)
      w[j] = u[i][j]* I[i]+ w[j];

/****** Write the Coefficients to the *_coefs File ******/


for(i=1; i≤M; i++)
   fprintf(outfile, "%f ", w[i]);

/****** Write the Name of the Input Face to *_coefs ******/

i = 0;
while(infilename[i] ≠ '\0')
      if (isalpha(infilename[i]))
         i++;
      else
         infilename[i] = 0;

strcpy(user,infilename);
fprintf(outfile, "%s\n", user);
free_matrix(u,1,N,1,M);

} /*end coefficients.c */
```

6.15 net_coefficients.c

```
/*********************************************************
*   Name:       net_coefficients.c
*
*
*   Description:  This program maps a test face onto the set of
*         eigenfaces and stores the KL coefficients in
*         train_coefs in a format the neural network can read.
*
*   Written by:  Pedro Suarez   (Originally recon.c)
*
*   Date:       24 July 91
*
*   Modified by:  Ken Runyon   (Chopped off reconstruction)
*
*   Date:       22 Jun 92
*
*   Modifications:  I decided we didn't need to actually reconstruct and
*         store a face.  I also made the stand alone program
*         into a module which is called by thesis.
*********************************************************/
#include <stdio.h>
#include <math.h>

void net_coefficients(dimension, num_coefs, infilename, outfile, classfile,
 num_class)
  int   dimension,
     num_coefs,
     num_class;

  char   infilename[];

  FILE   *outfile,
     *classfile;
{
FILE    *face1, *eigenin, *train, *face_avg;
int     i, j, l, N, M, atoi();
static int count = 0, exemplar = 0;
float     *vector(), **matrix(), *average_face, **u, *pedro, *reconface;
float     *w, *I;
char    filename[81], *strcpy(), user[8], ext[10];
static char user1[9],user2[9];

printf("\nPulling Coefficients for %s\n",infilename);

/****************** Set Up Files ***********************/

/*** Open Test Face ***/

if ((face1=fopen(infilename,"r")) == NULL){
 printf("I can't open the input file");
 exit(-1);
 }

/*** Open Avg Face ***/

if ((face_avg=fopen("avg_face.dat","r"))== NULL){
 printf("I can't open avg_face.dat.");
 exit(-1);
 }

/****** set up matrices ******/

N = dimension * dimension;
M = num_coefs;

u = matrix(1,N, 1,M);
pedro = vector(1, N);
average_face = vector(1, N);
reconface = vector(1, N);   /*** DO I NEED THIS? ***/
w = vector(1, M);
I =  vector(1, N);
```

```
/****** Initalize Matrices ******/

for(j=1; j≤M; j++)
   for(i=1;i<N;i++)
   w[j]=u[i][j]=I[i]=pedro[i]=reconface[i]=average_face[i]=0.0;

/********* Load the Test Face into the Pedro Vector ********/

 for(j=1;j≤N;j++)
   fscanf(face1,"%f\n", &pedro[j]);
 fclose(face1);

 /********* Load the Average Face into the Average_Face Vector ***/

for(j=1;j≤N;j++)
    fscanf(face_avg, "%f\n", &average_face[j]);
fclose(face_avg);

/****** Load the Eigenfaces into Matrix U ******/

train = fopen( "train.out","r");

for(j=1; j≤M; j++){
   fscanf(train, "%s\n", filename);
   eigenin = fopen(filename, "r");
   for(i=1;i≤N;i++){
      fscanf(eigenin,"%f\n",&u[i][j]);
   }
   fclose(eigenin);
}
fclose(train);

/****** Subtract the Average Face from the Test Face ******/

   for(i=1;i≤N;i++)
      I[i]= pedro[i] − average_face[i];

/****** Calculate the KL Coefficients ******/

for(j=1; j≤M; j++)
   for(i=1; i<N; i++)
      w[j] = u[i][j]* I[i]+ w[j];

/****** Write an exemplar number to the file ******/

fprintf(outfile, "%d   ", exemplar);
exemplar++;

/****** Write the Coefficients to the *_coefs File ******/

for(i=1; i≤M; i++)
   fprintf(outfile, "%f  ", w[i]);

/****** Write the desired outputs to the *_coefs File ******/

i = 0;
while(infilename[i] ≠ '\0')
      if (isalpha(infilename[i]))
         i++;
      else
         infilename[i] = 0;

strcpy(user1,infilename);

if(strcmp(user1,user2)){
   fprintf(classfile,"%8s\n",infilename);
   count++;
   strcpy(user2,user1);
   }

for(l=num_class;l≥1;l--){
   if(count==l)
      fprintf(outfile,"%f   ", 0.90000);
   if(count≠l)
      fprintf(outfile,"%f   ", 0.10000);
```

```
}

fprintf(outfile, "\n");
free_matrix(u,1,N,1,M);

} /* end coefficients.c */
```

6.16 display.c

```
/*************************************
*  display.c
*  converts a .gra file to rle and
*  displays it in openwindows using
*  xli.
*************************************/

int   dimension,
   num_pro;

char   filename[];

void display(dimension, filename,num_pro)
{

char      command[80];

static int   counter=0;

  sprintf(command, "%s%s%s", "cp ", filename,".gra temp.rec");
  system(command);
  system("float_gray temp.rec temp.red");
  switch(dimension){
    case 640:
     system("graytorle -o temp.rle 640 480 temp.red");
   break;
    case 128:
     system("graytorle -o temp.rle 128 128 temp.red");
   break;
    case 64:
     system("graytorle -o temp.rle 64 64 temp.red");
   break;
    case 32:
     system("graytorle -o temp.rle 32 32 temp.red");
   break;
    default:
   printf("I don't know what size the gra image is.");
   }
  system("rleflip -v -o hold.rle temp.rle");
  sprintf(command,"%s%s%s","mv hold.rle ", filename,".rle");
  system(command);

  sprintf(command,"%s%s%s","xli -quiet -zoom 300 -smooth -smooth ", filename, ".rle&");

/*  counter++;*/

  system(command);
/*  if(counter == num_pro) {
   counter = 0;
     system("rm *.red");
    system("rm temp.*");
     system("rm *.rle");
   }*/
 }
```

6.17 globals.h

```
/*
 * File:    globals.h
 * Created:    August 1992
 * By:      Kevin Gay
 *
 * Purpose:   Put all global variables and definitions in one place.
 *
 * Assumes:   vfc_lib.h is also included - all the vfc routines and
 *      vfc definitions are in vfc_lib.h.
 *
 * Modified:
 * By:
 * Why:
 */

#define PORT       "VFC_SVIDEO"
#define YUV_SIZE   VFC_NTSC_HEIGHT*VFC_YUV_WIDTH*2   /* 720 x 480 x 2bpp */
#define NTSC_SIZE    VFC_NTSC_WIDTH*VFC_NTSC_HEIGHT   /* 640 x 480 */
#define SM_WIDTH   32
#define SM_HEIGHT   32
#define SM_SIZE      SM_WIDTH*SM_HEIGHT
#define MOTION_THRESHOLD 3000

struct region
   {
   int      x;
   int      y;
   int      width;
   int      height;
   struct region   *next;
   };

struct hw_controls
   {
   VfcDev      *vfc_dev;
   int      colormap_offset;
   };

struct image_ptrs
   {
   u_char      *image1;
   u_char      *image2;
   u_char      *motion;
   };

struct head_ptrs
   {
   u_char         *head;
   struct head_ptrs   *next;
   };
```

6.18 gwind.c

```
/*****************************************************************
This routine takes an image by a guassian window.


written by:    Pedro F. Suarez
          29 July 91

Modified By:    Ken Runyon
     16 July 92

Modification:   Just made the c program a module to be called by
     thesis.c


*****************************************************************/
#include <stdio.h>
```

```
#include <math.h>
#define pi  3.1416
#define SQ(a)  a*a


void gwind(row,image_file)
  int   row;

  char   image_file[];
{
int   col = row,
  i, j, k, count,
  outval, temp_i, temp_j;

FILE *fin, *fout, *fo;
float   **p, **w, **matrix(), test,
  xmean = 16,
  xvar =10,
  ymean = 16,
  yvar = 14,
  normal,
  inval, max, min;

double  exp();

extern void rescale();

printf("\nGaussian Processing %s\n",image_file);

/*************** Set Up Files ********************************/

if ((fin=fopen(image_file,"r")) == NULL){
   printf("I can't open the input file");
   exit(-1);
  }


/******************** Allocate Memory To Matrices ***************/

p = matrix(0, row-1, 0, col-1);
w = matrix(0, row-1, 0, col-1);

/******************** Read The Input File Into The Matrix ********/

for(j=0; j<row; j++)
  for(i=0; i<col; i++)
    fscanf(fin, "%f\n",&p[i][j]);

fclose(fin);

/******************** Calculate the Gaussian Window ***************/

normal =  1/(2 * pi * xvar * yvar);

for(i=0; i<row; i++)
  for(j=0; j<col; j++){
    w[i][j] =  exp ((double) -0.5 *( SQ((i-xmean)/xvar)+ SQ((j-ymean)/yvar)));
    p[i][j] = p[i][j] * w[i][j];
}

rescale(p, row, col);

rescale(w, row, col);

/***************** Reopen the input file as an output file ***********/

if ((fout=fopen(image_file,"w")) == NULL){
   printf("I can't open the output file");
   exit(-1);
  }

/***************** Write the Result Back to the Input File ***********/

for(j=0; j<row; j++)
  for(i=0; i<col; i++)
```

```
            fprintf(fout, "%4.0f\n ", p[i][j]);
fclose(fout);

/************************ Store the gauss data into wind.dat **********/

/*fo = fopen("wind.dat", "w");

for(j=0; j<row; j++)
   for(i=0; i<col; i++)
      fprintf(fo, "%4.0f\n ", w[i][j]);
fclose(fo);*/

free_matrix(p,0, row-1, 0, col-1);
free_matrix(p,0, row-1, 0, col-1);

}
```

6.19 mdct.c

```
/*

mdct.c

   This program takes the 2-D Discrete Cosine Transform (2D-DCT) and the inverse (2D-IDCT) of an NxNx8 bit image.
The code is designed to work with any square image, but could be easily modified to work with any size image.

   The program is designed to do a symmetric transform if so desired, This option merely flips and folds the image to 2Nx2N
pixels then takes the DCT. This is not necessary, but can be interesting.  I can't think of any practical reason to do this.  It
was thought to be necessary to have an even function when this research started.

nrutils.o must be included in the make file.

The Cosine Kernel matrix must be in the same folder.  This is generated by CreateCMatrix.c.

Written by Jim Goble, June 1991
Disclaimer -

Copyrighted by the Air Force Institute of Technology and by James R. Goble.  May be used for any non-profit application
without permission.  This code is presented as is, and no claims are made as to suitability for other applications.  It is not
guaranteed to be error free.

Modified by:  Ken Runyon, Sep 92

1.  Redid the structure to make the program a module to be called by train and recognition programs.

2.  Got rid of prompts for image size, non-symmetric/symmetric transform, and inverse transform.

3.  Also pass infile as a parameter, and hardcode outfile as train_coefs.

4.  Shortened resulting dct component matrix to just take an 8 x 8 rectangle of the fist coefficients.

5.  Added the user name to the output to create the train_coefs file.


*/

#include <stdio.h>
#include <math.h>
#define SQ(A)   (A*A)

void dc_transform(N,ifname,DCTfile,feature_dimensions)

FILE *DCTfile;
char ifname[];
int N,feature_dimensions;

{
FILE *Cfile, *Ifile;
char user[8], answer, answer2;
float **matrix(); /* prototype call to nrutil routine */
float **CosMat, **ICosMat, **TranMat, **TempMat; /* reserve pointers for my matrices */
float **ImMat;  /* reserve pointer for my image matrix */
int i,j,k, NN, N2, Z, valid, features;
```

```
printf("%d\n%s\n%d\n",N,ifname,feature_dimensions);

/*
     Get the input image.  This routine also checks for file I/O errors.

*/
  if((Ifile=fopen(ifname,"r"))==NULL)
     {
     printf("File does not exist! %s",ifname,"\n");
     printf("\n");
     } /* end of if */

/* Open the Cosine Matrix CosMat.dat.  The matrix is always kept in this file.  It is assumed that the proper size matrix is
stored here.  If not, generate a new one with CreateCMatrix.  Code checks for file errors, but not for incorrect matrix size.
C, being the wonderful excuse for a language that it is, will run the program with whatever happens to be in memory if you
do not have the proper matrix size.

*/
if((Cfile=fopen("CosMat.dat","r"))==NULL)
   {
   printf("Can't open file! CosMat.dat\n Run create.");
   printf("\n");
   exit(0);
   } /* end of if */

/*    Set parameters for DCT */

     NN = N - 1;
     N2 = N-1;
     Z = N2;
printf("N = %d\n NN = %d\n N2 = %d\n Z = %d\n",N,NN,N2,Z);
     ImMat = matrix(0,N2,0,N2);
     ICosMat = matrix(0,N2,0,N2);
     CosMat = matrix(0,N2,0,N2);
     TranMat = matrix(0,N2,0,N2);
     TempMat = matrix(0,N2,0,N2);

/* Initialize the Appropriated Matrices */
for(i=0; i<N2; i++)
   for(j=0; j<N2; j++)
     TranMat[i][j]=TempMat[i][j]=0.0;

     /* Read in the cosine matrix from disk. */
     printf("!!!  Opening and Reading Cosine Matrix!:\n ");
     Cfile=fopen("CosMat.dat","r");
     for (i = 0; i < N2; i++)
     for (j = 0; j < N2; j++)
     {
     fscanf(Cfile,"%f",&CosMat[i][j]);
      ICosMat[j][i] = CosMat[i][j];
     }

     fclose(Cfile);
     printf("!!!  Finished Reading Cosine Matrix!:\n ");

     printf("!!!  Opening and Reading Image Matrix!:\n ");
     for (i = 0; i < N2; i++)
     for (j = 0; j < N2; j++)
       {
       fscanf(Ifile,"%f",&ImMat[i][j]);
     }
     printf("Taking DCT !:\n ");

  fclose(Ifile);

  printf("!!!  Finished Reading Image Matrix!:\n ");

  /* Do DCT */

  for (i = 0; i < N2; i++)
     {
     for (j = 0; j < N2; j++)
       {
```

```
        for (k = 0; k ≤ N2; k++)
          {
          TempMat[i][j] = TempMat[i][j] + CosMat[i][k]*ImMat[k][j];
          }
        }
      }
   for (i = 0; i ≤ N2; i++)
      {
      for (j = 0; j ≤ N2; j++)
        {
        for (k = 0; k ≤ N2; k++)
          {
          TranMat[i][j] = TranMat[i][j] + TempMat[i][k]*ICosMat[k][j];
          }
        }
      }
   printf("Finished Taking DCT !:\n ");


printf("Writing Output!:\n ");

/* Now output the results to the file named above. */

 printf("Writing Output!:\n ");
/* Now output the results to the file named above. */
for (i = 0; i≤ feature_dimensions; ++i) {
      for (j = 0; j ≤ feature_dimensions; ++j) {
           fprintf(DCTfile,"%f\t", TranMat[i][j]);
           }    /* end of n for loop */
      } /* end of m for loop */

/****** Write the Name of the Input Face to *_coefs ******/

i = 0;
while(ifname[i] ≠ '\0')
      if (isalpha(ifname[i]))
         i++;
      else
         ifname[i] = 0;

strcpy(user,ifname);
fprintf(DCTfile, "%s\n", user);
system("ls *.gra");

} /* end of DCT.c */
```

6.20 rescale.c

```
/*****************************************************************************
   NAME: rescale
   DESCRIPTION: This routine scales the matrix into the 0 - 255 range
*****************************************************************************/

#include <stdio.h>
#include <math.h>

void rescale(output, row, col)

float **output;
int row, col;
{
int    NEW_MAX = 255,
   NEW_MIN = 0,
   i, j,
   count;

float  min,
   max;

/** Check for the max and min value in the data **/
```

```
min=max=output[0][0];
count=0;

for(j=0; j<row; j++)
      for(i=0; i<col; i++){
        if(output[i][j]>max)
          max=output[i][j];
        if(output[i][j]<min)
          min=output[i][j];
       count++;
   }

/*** Now translate data and write to output file ***/

for(j=0; j<row; j++)
  for(i=0; i<col; i++){
    output[i][j] = ((output[i][j]-min)*(NEW_MAX-NEW_MIN)/(max-min) + NEW_MIN);
 }
  }
```

6.21 grab.c

```
/*
 * File:   grab.c
 * Created:    8 July 1992
 * By:      Kevin Gay
 *
 * Purpose:   This code is intended to collect in images in a loop num_loop.
       Each pass thru the loop takes in YUV image data,
      converts the data to 8-bit gray, square pixel data, and
      save the 8-bit data in a file labelled with person's name.
      [Note: z_binarize_gra allows image to be binarized
      just change z_store_image var gray_data to bin_data]
      Number of loops (num_loop) and person's name (person)
      are entered during execution.

 Assumes:   Signal coming in S-Video port, NTSC format
 */

#include <stdio.h>
#include <sys/types.h>
#include "vfc_lib.h"
#include "globals.h"

extern struct hw_controls   *z_set_vfc_hw();
extern u_char          *z_grab_gra();
extern u_char          *z_reduce();
extern int          z_store_image();

int   dimension;

void grab(dimension)
{
  u_char          *gray_data, *sm_data;
  struct hw_controls   *hw_ptrs;
  char          filename[30];
  int          S_WIDTH = dimension,
              S_HEIGHT = dimension,
              S_SIZE = S_WIDTH * S_HEIGHT;


  /*************** BEGIN IMAGE GRABBING LOOP ****************/

  printf("\nGrabbing the Image\n");

  hw_ptrs = (struct hw_controls *)z_set_vfc_hw();

  /*
   * Read in a 8 bit gray image
   */
```

```
  gray_data=(u_char *)z_grab_gra(hw_ptrs);

  if(gray_data == NULL)
    {
    printf("ptr is null\n");
    exit(1);
    }

  /*
   * Create a reduced image from the 8 bit gray data.
   */

  sm_data=(u_char *)z_reduce(gray_data,VFC_NTSC_WIDTH,VFC_NTSC_HEIGHT,
          S_WIDTH,S_HEIGHT);
  if(sm_data == NULL)
    {
    printf("small ptr is null\n");
    exit(1);
    }

    /*
     * Create name for image data and store in file.
     */

  sprintf(filename,"%s","user.red");

  if(z_store_image(sm_data, filename, S_SIZE) < 0)
    fprintf(stderr,"Unable to write %s to file\n",filename);

  /************** END IMAGE GRABBING LOOP ************************/

    /*
   * Now destroy the hardware and free the memory.
   */
  vfc_destroy(hw_ptrs->vfc_dev);
  free(sm_data);
    free(gray_data);
  free(hw_ptrs);
} /*** end grab.c ***/
```

6.22 k_nearest.c

```
/*********************************************************
*   Name:       k_nearest.c
*
*   Description:  This program finds the k nearest neighbors for a given
*         test image where k is the number of prototypes for
*         each person in the training set.  The nearest neighbors
*         are rank ordered by euclidean distance.  Scores for
*         each person found are calculated by summing weighted
*         values for each position in the ranked nearest neighbor
*         list.  The winner is the person with the highest score.
*
*   Written by:  Ken Runyon
*
*   Date:       11 Sep 92
*
*********************************************************/
#include <stdio.h>
#include <math.h>
#define MIN_ARRAY_SIZE   20
#define   numberblocks   300
#define START       100000
#define SQ(A)   (A*A)

typedef char *string;

void k_nearest(num_protos,num_coefs,num_train_faces)
  int   num_protos,
      num_coefs,
      num_train_faces;
{
```

```
FILE    *ftest,
   *ftrain,
   *fout;

float    distance,
   temp,
   bottom,
   *average,
   *sd,
   *vector();

int    j,
   i,
   k,
   begin,
   max;

char    name[20],
   command[80],
   filename[30];

double sqrt();

struct    block
   {
   float    *feature_vec;
   char     name[20];
   float    distance;
      };

struct    rank_array
   {
   char    name[20];
   int    score;
   float    distance;
      };

struct block atest[1];
struct block btrain[numberblocks];
struct rank_array score_pad[MIN_ARRAY_SIZE];

printf("\nFinding the Nearest Neighbors\n");

/*****************Open the Files *************/

 if ((ftrain=fopen("train_coefs","r"))== NULL){
  printf("I can't open the train coefficients file");
  exit(-1);
  }

if ((ftest=fopen("test_coefs","r"))== NULL){
  printf("I can't open the test coefficients file");
  exit(-1);
  }


/****** Initialize btrain block ******/

for(i=1; i<=num_train_faces; i++){
   btrain[i].feature_vec=vector(1, num_coefs);
   btrain[i].distance = 0.0;
}

/****** Initialize atest block ******/

   atest[0].feature_vec=vector(1, num_coefs);
   atest[0].distance = 0.0;

/****** Create a Couple of Vectors ******/

average=vector(1, num_coefs);
sd=vector(1, num_coefs);

/****** Initialize those Vectors ******/
```

```
for(i=1;i≤num_coefs; i++)
   average[i]=sd[i]=0.0;

/****** Read the Test Coefficients into the atest Matrix ****/

for(j=1; j≤num_coefs; j++) {
   fscanf(ftest, "%f", &temp);
   atest[0].feature_vec[j]=temp;
   }

/****** Then Read in the Name that Belongs to those Coefficients ******/

fscanf(ftest, "%s\n", atest[0].name);


/****** Read the Training Database into btrain Matrix ******/

for(i=1; i≤num_train_faces; i++){
   for(j=1; j≤num_coefs;j++) {
      fscanf(ftrain, "%f", &temp);
      btrain[i].feature_vec[j]=temp;
   }
   fscanf(ftrain, "%s\n", btrain[i].name);
}

/****** Statistically Normalize ******/

for(j=1; j≤num_coefs; j++)
   average[j]+=atest[0].feature_vec[j]/(num_train_faces + 1);

for(j=1; j≤num_coefs; j++)
   for(i=1; i<num_train_faces; i++)
     average[j]+=btrain[i].feature_vec[j]/(num_train_faces + 1);


for(j=1; j≤num_coefs;j++)
  sd[j] += (atest[0].feature_vec[j] − average[j])*
(atest[0].feature_vec[j] −average[j]);

for(j=1; j≤num_coefs;j++)
   for(i=1;i≤num_train_faces;i++)
      sd[j] += (btrain[i].feature_vec[j] − average[j]) *
(btrain[i].feature_vec[j] − average[j]);

for(j=1;j≤num_coefs;j++){
   sd[j] = sqrt( (double) sd[j]);
   sd[j] = ((double) 1/num_train_faces) * sd[j];
}

/****** Calculate Euclidean Distance to Each Prototype ******/

for(i=1; i≤num_train_faces; i++){
   temp=0;
   for(j=1; j≤num_coefs;j++) {
      temp = (atest[0].feature_vec[j] − btrain[i].feature_vec[j]) *
(atest[0].feature_vec[j] − btrain[i].feature_vec[j]) + temp;
      btrain[i].distance = sqrt ( (double) temp);
   }
}

/****** Store the k-nearest neighbors rank ordered by distance ******/

for(j=1; j≤num_protos; j++){
   temp = START;
   for(i=1; i≤num_train_faces; i++)
      if((btrain[i].distance < temp)&&(btrain[i].distance > bottom)){
       temp = btrain[i].distance;
       strcpy(score_pad[j].name, btrain[i].name);
       score_pad[j].distance = btrain[i].distance;
      }
   bottom = temp;
}

/****** Assign Weights for Each Position in the Rank_Ordered Matrix *******/
```

```
for(j=1; j≤num_protos; j++){
   score_pad[j].score = num_protos+1 − j;
}

/********** Print the Rank **********/

/*printf("\nLooking for : %s\n\n", name);
for(j=1; j≤num_protos; j++)
printf("\n%d\t%8s\t%4.0f\t%d",j,score_pad[j].name,score_pad[j].distance,
score_pad[j].score);*/

/****** Tally the Scores ******/

for(i=1; i≤num_protos; i++){
   strcpy(name,score_pad[i].name);
   begin = i+1;
   for(j=begin; j≤num_protos; j++){
      if(!strcmp(score_pad[j].name, name))
         score_pad[i].score += score_pad[j].score;
   }
}

/********** Find out who the user is **********/

strcpy(name,(char *)getenv("USER"));

/********** Print the Score **********/

printf("\n\nLooking for : %s\n", name);
for(j=1; j≤num_protos; j++)
printf("\n%d\t%8s\t%4.0f\t%d",j,score_pad[j].name,score_pad[j].distance,
score_pad[j].score);

/****** Find the Max Score ******/

max = 1;
temp = score_pad[max].score;
for(i=1; i≤ num_protos; i++)
   if(temp<score_pad[i].score){
      max=i;
      temp=score_pad[i].score;
   }

/********** Pull Up the Image of the Nearest Neighbor **********/

sprintf(filename,"%s%s%d","training_images/",score_pad[max].name,1);
/*display(32,filename,1);*/


/****** Signify the Person Recognized and Display the Image ******/

if(!strcmp(name,score_pad[max].name))
   printf("\n\nLogged in as %s\n",name);
else {
   printf("\n\nYou've been recognized as %s\n",score_pad[max].name);
   /*system("logout");*/
}

} /*** end k_nearest ***/
```

Bibliography

1. Cottrell, Garrison W. and Janet Metcalfe. "EMPATH: Face, Emotion and Gender Recognition Using Holons," *Unpublished* (1991).
2. Farahati, *et al.* "Real-time recognition using novel infrared illumination," *Optical Engineering*, *31*:1658–1662 (August 1992).
3. Fleming, Michael K. and Garrison W. Cottrell. "Categorization of Faces Using Unsupervised Feature Extraction," *IEEE International Joint Conference on Neural Nerworks*, 2:65–70 (1990).
4. Gay, Kevin P. *Autonomous Face Recognition.* MS thesis, AFIT/GE/ENG/92D. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, September 1992.
5. Goble, James R. *Face Recognition Using the Discrete Cosine Transform.* MS thesis, AFIT/GE/ENG/91D-21. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1991.
6. Hayden D. Ellis, *et al*, editor. *Aspects of Face Processing.* The Netherlands: Martinus Nijhoff Publishers, 1986.
7. Kirby, M. and L. Sirovich. "Applications of the Karhunen-Loève procedure for the characterization of human faces," *IEEE Transactions PAMI, 12* (1990).
8. Krepp, Dennis L. *Face Recognition with Neural Networks.* MS thesis, AFIT/GE/ENG/92D. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1992.
9. Lambert, Lawrence C. *Evaluation and Enhancement of the AFIT Autonomous Face Recognition Machine.* MS thesis, AFIT/GE/ENG/87D-35. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987.
10. Payne, *et al.* "Backpropagation Neural Networks for Facial Verification Update," *Los Alamos National Laboratory* (1992 Unpublished).
11. Robb, Barbara C. *Autonomous Face Recognition Machine Using a Fourier Feature Set.* MS thesis, AFIT/GE/ENG/87D-35. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987.
12. Routh, Richard L. *Cortical Thought Theory: A Working Model of the Human Gestalt Mechanism.* PhD dissertation, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1985.
13. Ruck, Dennis W. *Characterization of Multilayer Perceptrons and their Application to Multisnesor Automation Target Detection.* PhD dissertation, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1990.
14. Russell, Robert L. *Performance of a Working Face Recognition Machine Using Cortical Thought Theory.* MS thesis, AFIT/GE/ENG/85D. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1985.
15. Sander, David D. *Enhanced Autonomous Face Recognition Machine.* MS thesis, AFIT/GE/ENG/89D-19. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1989.

16. Smith, Edward J. *Development of an Autonomous Face Recognition Machine.* MS thesis, AFIT/GE/ENG/86D-36. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1986.

17. Suarez, Pedro F. *Face Recognition with the Karhunen-Loeve Transform.* MS thesis, AFIT/GE/ENG/91D-54. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1991.

18. Turk, Matthew A. and Alex P. Pentland. "Eigenfaces for Recognition," *Journal of Cognitive Neuroscience*, 1–28 (September 1990).

19. Turk, Matthew A. and Alex P. Pentland. "Recognition in Face Space," *SPIE Intelligent Robots and Computer Vision IX: Algotithms and Techniques, 1381*:43–54 (1990).

Vita

Captain Kenneth R. Runyon was born on January 9, 1961 in Logan, West Virginia. He graduated from Man High School in Man, West Virginia in 1979. Capt. Runyon entered the Air Force in September, 1979 as an Instrumentation Mechanic. He served five years at Wright Patterson AFB, Ohio with the Flight Dynamics Laboratory, Air Force Wright Aeronautical Laboratories. He entered the Airman's Education and Commissioning Program in March, 1985 and completed a Bachelor of Science degree in Systems Engineering at Wright State University in Dec, 1987. He served three years with the Space Systems Division at Los Angeles AFB, California before entering the School of Engineering, Air Force Institute of Technology in June, 1991. He is married to Lisa (Adkins) Runyon of Robinette, West Virginia and has two children: Anna Marie, age 11, and Sarah Lynn, age 1.

Permanent address: P.O. Box 339
Man, West Virginia 25635

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1992	Master's Thesis

4. TITLE AND SUBTITLE
Automated Face Recognition System

5. FUNDING NUMBERS

6. AUTHOR(S)
Kenneth R. Runyon, Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GE/ENG/92D-33

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Maj Rodney Winter
DIR/NSA,R221
9800 Savage Rd
Ft Meade, MD 20755-6000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

In this thesis three variations of an end-to-end face recognition prototype system are developed, implemented and tested. Each version includes real-time image collection, automated segmentation, preprocessing, feature extraction, and classification. The first version uses a Karhunen Loéve Transform (KLT) feature extractor and a K-nearest neighbor classifier. Version two uses the same feature set but utilizes a multilayer perceptron neural network with a back propagation learning rule. Finally the third version uses a Discrete Cosine Transform as the feature extractor and the K-nearest neighbor as the classifier. Only the KLT versions of the system were tested. The tests were based on three image sets, each collected over multiple days to analyze the effect on recognition accuracy of variations in both the image collection environment and the subjects over time. The first set consisted of 23 Subjects and was taken over a two day period. The second set consisted of four users and was taken over a seven day period. Finally, the third set consisted of 100 images of a single subject collected over several weeks.

14. SUBJECT TERMS	15. NUMBER OF PAGES
face recognition, karhunen loéve transform, discrete cosine transform, k-nearest neighbor, neural networks, backpropagation, user verification	118
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL